RUPRECHT-KARLS-
UNIVERSITÄT HEIDELBERG

HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

Effiziente Berechnung von Kookkurrenzwerten unter Verwendung von öffentlichen Suchmaschinen-Indizes

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.) im Studiengang
Medizinische Informatik

vorgelegt von

Johannes Merz, 13. Februar 2012

Referent:	Prof. Dr. Daniel Pfeifer
Korreferent:	Prof. Dr. Rolf Bendl
Betreuer:	Dipl.-Inform. Med. Martin Wiesner

Abstract

Im Kern der Bachelorarbeit steht die Berechnung der semantische Nähe zwischen zwei medizinischen Konzepten. Es ist das primäre Ziel, ein Verfahren zu implementieren, welches durch Verwendung von Suchmaschinen-Indizes Signifikanzwerte für die Konzeptrelationen von sogenannten Wiki-Graphen berechnet [Hei11]. Derartige Datenstrukturen finden in sogenannten Health Recommender Systemen (HRS) zur Berechnung von Empfehlungen Anwendung ([WP10], [Hei11]).

Ein besonderes Augenmerk liegt auf der Skalierbarkeit der zu entwickelnde Softwarekomponente. Da eine sehr hohe Anzahl an zu verarbeitenden Konzeptrelationen, d.h. Kanten im Wiki-Graphen, vorliegt und die Antwortzeit von öffentlichen Suchmaschinen-APIs limitiert ist, wurde eine verteilte Berechnung mit Hilfe eines Rechenclusters konzipiert.

Als primäres Ergebnis wurde ein Java-API entwickelt, welches für eine beliebige medizinisch geprägte Fachsprache Signifikanzwerte auf Basis der Suchmaschine BING berechnet. Der Verteilungsalgorithmus, welcher mit der Hilfe des JPPF-Frameworks realisiert wurde, ermöglicht unter idealen Bedingungen eine zeiteffiziente Berechnung von Kantengewichten. Zur Berechnung der Signifikanzwerte wurden mehrere Strategien wie z.B. das Jaccard-Maß, Log-Liklihood, sowie das Maß der Mutual Information realisiert. Diese sind aus der Literatur zum Forschungsgebiet `Text Mining` beschrieben.

Als sekundäres Ergebnis stehen die mit Hilfe der Suchmaschine Bing automatisch berechneten Signifikanzwerte. Diese Werte liefern bisher jedoch keine zufriedenstellenden bzw. ausagekräftigen Kantengewichte. Andere Suchmaschinen wurden im Verlauf der Arbeit verwendet, konnten jedoch aus Kostengründen nicht ernsthaft für eine komplette Analyse eingesetzt werden.

*Danksagung

Bedanken möchte ich mich bei meinem Referenten: Herrn Prof. Dr. Daniel Pfeifer für die fachliche Unterstützung. Er hatte jederzeit und für jedes Problem ein offenes Ohr.

Ein ganz besonderer Dank geht an meinen Betreuer Herrn Martin Wiesner: Für die engagierte, professionelle Betreuung und für die enorme Zeit die er für die Besprechungstermine geopfert hat. Vor allem aber für seinen Einsatz beim Korrekturlesen und das geduldige Ertragen meiner Rechtschreibfehler!

Ein dickes Dankeschön geht auch an meinen guten Freund Andreas Fetzer, der mich gerade in der Schlussphase so sehr unterstützt und an Tagen, an denen nichts voranging, aufgebaut und motiviert hat.

Inhaltsverzeichnis

Glossar

Abfrage
Abfrage bezeichnet eine Anfrage an eine *Suchmaschine* `M` und deren korrespondierenden *Index* `i` für einen bestimmten *Term* `e`.

Gesundheitsakte
Eine Gesundheitsakte (personal health record) ist eine Akte, in welcher Gesundheitsdaten von einem einzelnen Benutzer selbst verwaltet werden können. [Wik11a].

Index
≫*Der Index der Suchmaschine beinhaltet alle bisher bekannten und von der Suchmaschine selbst verarbeiteten Seiten. Verarbeitet bedeutet hier, daß die Suchmaschine die betreffende Seite gefunden, "gelesen"und die Kerniformation der Seite dauerhaft gespeichert hat* ≪ [Pre10].

Konzept
Ein Konzept bezeichnet in dieser Arbeit einen Wikipedia-Eintrag welcher im *Wiki-Graph* durch einen Knoten repräsentiert wird.

Konzeptrelation
Repräsentiert die Beziehung (Kante) zwischen zwei Konzepten (Knoten) im *Wiki-Graphen*.

Kookkurrenz
Kookkurrenz bezeichnet das gemeinsame Auftreten zweier Wortformen in einem lokalen Kontext (vgl. [HQW06]).

Signifikanzwert
Wert, der die Signifikanz der Kookkurrenz von zwei *Konzepten* beschreibt.

Suchmaschine
Eine Suchmaschine ist ein Informationssystem zur Recherche von Dokumenten und ähnlichen Informationsartefakten, die auf einem Computer oder innerhalb eines Computernetzwerks wie z.B. dem World Wide Web gespeichert sind. [Wik11b].

Suchtreffer
Auftreten eines *Terms* im *Index* einer *Suchmaschine*. Ein Suchtreffer

`t` erzeugt für eine Anfrage `q` eine Menge von relevanten Dokumenten `d`.

Term
Ein Term definiert sich als Wort oder Wortkombination für die Anfragemenge `Q` an einen Suchmaschinenindex `i`. Synonym: Suchterm.

Textkorpus
Der Ausdruck Textkorpus oder Korpus bezeichnet generell eine Sammlung von schriftlichen Texten in einer bestimmten Sprache (vgl. [Wik12]).

Wiki-Graph
Gerichteter, typisierter, benannter, zyklischer Graph der auf den Strukturmerkmalen Kategorien, Artikel, ICD-10-Codes, Weiterleitungsseiten und Links der Wikipedia aufgebaut ist.

Wortform
Flektierte Form eines Wortes. Jede Form, die ein Wort durch Beugung, Ableitungen, Zusammensetzungen annhemen kann.

1 Einführung

1.1 Motivation

Das Projekt TULUM an der Hochschule Heilbronn beschäftig sich mit der Entwicklung eines Empfehlungsdienstes für Nutzer von *Gesundheitsakten*. Im Kern der aktuellen Forschung stehen sogenannte Health Recommender Systeme (HRS). Solche Systeme nutzen automatisierte Verfahren, welche ausgehend von der textbasierten Eingabe eines Benutzers semantisch nahe Fachartikel bzw. relevante Informationen ermitteln und empfehlen [Hei11]. Da der Kontext eines HRS sich spezifisch mit der Gesundheitsdomäne befasst, erfolgt die Berechnung der semantischen Nähe mit Hilfe von graphenbasierten Datenstrukturen, sogenannten Gesundheitsgraphen [Tri10]. Zum Zeitpunkt der Bachelorarbeit erfolgte die Berechnung der semantischen Nähe der medizinischen Konzepte (d.h. den Kanten des Gesundheitsgraphen) auf der Basis einer rudimentären Heuristik. Diese Heuristik weist vorhandenen Konzeptrelationen statisch definierte Grundgewichte je nach Kantentyp zu. Da dies jedoch eine relativ vereinfachte semantische Annäherung an eine fundiertere, reale signifikante Kookkurrenz ist, besteht der Bedarf einer semantisch validen Berechnung, die auf dem Forschungsgebiet `Text Mining` aufbaut (vgl. [WP10]).

1.2 Ziele

Da im Kern der Bachelorarbeit die Verbesserung der Berechnung der semantische Nähe zwischen zwei medizinischen Themenkonzepten steht, ist es das primäre Ziel, ein Verfahren zu implementieren, das durch Verwendung von Suchmaschinen-Indizes Signifikanzwerte für die Konzeptrelationen des *Wiki-Graphen* berechnet. Dabei soll ein hohes Augenmerk auf die Skalierbarkeit der zu entwickelnden Softwarekomponente gelegt werden. Folgende Zielkriterien werden an das API gestellt:

1. Konzeption und Implementierung einer Softwarekomponente zur automatisierten Berechnung der Signifikanzwerte von Konzeptrelationen.
2. Bereitstellung eines Verteilungsalgorithmus zur zeiteffizienten Berechnung einer hohen Anzahl an Konzeptrelationen.
3. Definition eines Suchmaschinen unabhängiges API zur Kookkurrenzberechnung
4. Bereitstellung einer Softwarekomponente, welche Graph-Instanzen anhand der berechneten Signifikanzwerte gewichten kann.

2 Grundlagen

In den folgenden Abschnitten werden die Grundlagen erläutert, die zum besseren Verständnis der vorliegenden Bachelorarbeit beitragen. Dazu gehört die Einführung in die Datenstruktur des *Wiki-Graphen* (Kapitel 2.1), des Begriffs "Kookkurrenz", sowie statistische Signifikanzmaße (Kapitel 2.2), dem Web als Korpus (Kapitel 2.2.2), verwendete Werkezuge (Kapitel 2.3) und API basierte Nutzung von Suchmaschinen (Kapitel 2.4).

2.1 Datenstruktur des Wiki-Graphen

Das Konzept bzw. die Datenstruktur "*Wiki-Graph*"bildet die Basis des Empfehlungssystems TULUM. Es handelt sich dabei um einen gerichteten, typisierten, benannten, zyklischen Graphen, der aus den Strukturmerkmalen der Wikipedia aufgebaut ist. Dieser Graph dient zur Abbildung der Beziehungen medizinischer Konzepte und gibt Aufschluss über deren semantische Nähe. Der folgende Abschnitt beschreibt den Aufbau und die einzelnen Elemente, die im Wiki-Graphen abgebildet werden.

Die Strukturmerkmale, die im Graphen abgebildet werden, sind Artikel, Kategorien, Weiterleitungsseiten, ICD-10-Codes und Links. Jeder Knoten des Graphen repräsentiert ein Konzept der Wikipedia. Der verknüpfte Wikipedia-Artikel besteht zum größten Teil aus Fließtext und enthält den rein enzyklopädischen Teil. Kategorien teilen die einzelnen Artikel in Themenbereiche ein, welche einer Hierarchie (Oberkategorie, Unterkategorie, Artikel) unterworfen sind. Die so entstehende Baumstruktur gibt Aufschluss über die Themenbeziehungen innerhalb dieser Hierarchie. Weiterleitungen sind Artikel ohne Inhalt, die direkt auf einen anderen Artikel veweisen. Sie werden für alternative Begriffe, die dasselbe Konzept beschreiben, Sammelartikel und phonetische Schreibweisen verwendet. Alle drei Anwendungsfälle der Weiterleitungen weisen auf eine sehr starke Ähnlichkeit zwischen dem Quell- und Zielartikel hin, die als Synonyme interpretiert werden können (nach [Tri10], S.27 Ab.2). ICD-10-Codes werden in Wikipedia mit Hilfe einer Vorlage eingebunden. Eine Vorlage ist eine Schablone, welche Darstellungs- und Inhaltskonventionen vorgibt. Die deutsche Vorlage für den ICD-10-Code ist die `Infobox ICD`, welche dann den Code und Beschreibungen der entsprechenden Krankheit oder verwandter Krankheiten enthält. Diese vier Strukturelemente (Artikel, Kategorien, Weiterleitungsseiten, ICD-10-Codes) stellen im *Wiki-Graph* unterschiedliche Knotentypen dar. In den Knoten werden die thematisierten Konzepte abgebildet, die in Wikipedia durch Seiten (Artikel, Kategorien und Weiterleitungsseiten) repräsentiert werden. Ein Konzept kann in mehreren Repräsentationsformen auftreten, wie zum Beispiel Artikel und Kategorie für den Knoten „Dickdarm" (siehe Abbildung 1). Damit ein Konzept nur einmal durch einen Knoten vertreten wird, sind die Knoten mehrfach typisierbar und durch einen im gesamten Graphen eindeutigen Konzeptnamen benannt. Anhand der weiterführenden Links auf den Wikipedia-Seiten werden die einzelnen Knoten durch Kanten verbunden. Eine Kante ist daher immer dann im

Graphen vorhanden, wenn ein Konzept auf ein anderes Konzept verweist. Für die Kanten wurden folgende Assoziationsarten aus der Wikipediastruktur verwendet:

1. Artikel auf andere Artikel
2. Kategorien auf Artikel
3. Kategorie auf andere Kategorie
4. Weiterleitungsseiten auf Artikel
5. Artikel auf die in der `Infobox ICD` jeweils enthaltenen ICD-10 Codes

Diese fünf Varianten bilden die Kantentypen des WikiGraphen und sorgen für eine eindeutige Identifizierung der Verweise. In Abbildung 1 wird dies durch den Verweis des Knoten „Organ als Thema" auf den Knoten „Dickdarm" verdeutlicht. Durch die Kantentypisierung kann festgestellt werden, dass es sich um einen Verweis von einer Kategorie auf eine andere Kategorie handelt. Der WikiGraph kann Zyklen enthalten, wie zum Beispiel beim Knoten „Dickdarm", der zwei Knotentypen (Kategorie und Artikel) besitzt, wobei die Kategorie auf den gleichnamigen Artikel verweist (vgl. [Tri10]).

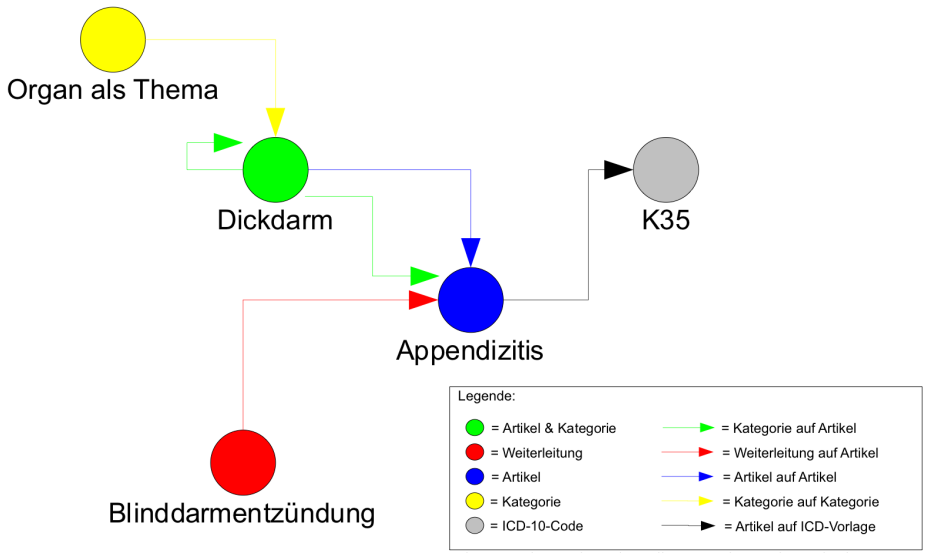

Abbildung 1: Fiktiver WikiGraph mit allen Strukturmerkmalen ([Tri10], S.37)

2.2 Kookkurrenz

Das gemeinsame Auftreten zweier *Wortformen* innerhalb eines definierten Textfensters wird als *Kookkurrenz* bezeichnet (vgl. [HQW06]). Ein Textfenster kann dabei ein Textabschnitt (z.B. ein Absatz), eine Seite, ein Dokument oder ein Fenster fester Länge sein. Treten zwei *Wortformen* aus statistischer Sicht (auffallend) häufig auf, ist dies ein starkes Indiz für einen semantischen Zusammenhang. Solche *Kookkurrenzen* werden auch signifikante Kookurrenzen genannt. Durch das Zählen von *Kookkurrenzen* und der Häufigkeit der einzelnen *Wortformen* über eine größere Sammlung von Texten (*Textkorpus*) kann nun die Stärke der Zusammengehörigkeit von *Wortformen* mittels eines Signifikanzmaßes berechnet werden.

"Aufgabe eines Maßes für die statistische Signifikanz einer Kookkurrenz ist es, einem Paar von Wortformen (wie z.B Polizei - verhaftet) eine Zahl zuzuordnen, welche die Signifikanz dieser Kookkurrenz beschreibt."([HQW06],S.137).

2.2.1 Signifikanzmaße

In den anschließenden Abschnitten werden die Signifikanzmaße Jaccard, Mutual Information, und Log-Likelihood Ratio vorgestellt.

Jaccard

Der Jaccard-Index ist ein Ähnlichkeitsmaß für Mengen, der einen Wertebereich zwischen [0,1] besitzt. Um den Jaccard-Index zweier Mengen A und B zu berechnen, teilt man die Schnittmenge ($|A \cap B|$) durch die Vereinigungsmenge ($|A \cup B|$). Zur Berechnung der Signifikanz von *Kookkurrenzen* wird für die Schnittmenge die Anzahl k_{ij} der *Wortformen*, die innerhalb eines definierten Textfensters gemeinsam im *Textkorpus* auftreten (Anzahl der Kookkurenzen) verwendet. Die Vereinigungsmenge ergibt sich aus der Anzahl k_i der ersten Wortform t_i plus der Anzahl k_j der zweiten Worform t_j minus der Schnittmenge k_{ij} (vgl.[HQW06],S.213).

$$sig_{jaccard}(t_i, t_j) = \frac{k_{ij}}{k_i + k_j - k_{ij}} \tag{1}$$

Mutual Information

Das Signifikanzmaß "Mutal Information" nach Fano (1961) vergleicht die beobachtete Wahrscheinlichkeit $P(x, y)$ mit der erwarteten Wahrscheinlichkeit $P(x)P(y)$ für das Auftreten zweier *Wortformen*. Sie ist definiert als:

$$I(x, y) = log_2 \frac{P(x, y)}{P(x)P(y)} \tag{2}$$

Ist die beobachtete Wahrscheinlichkeit deutlich größer als die zu erwartende Wahrscheinlichkeit ($I(x, y) >> 0$), dann gibt es einen semantischen Zuammenhang zwischen den *Wortformen* x und y. Wenn es keine Beziehung zwischen den beiden *Wortformen* gibt, dann ist $P(x, y) \approx P(x)P(y)$ und damit wäre $I(x, y) \approx 0$. Schließen sich die zwei *Wortformen* in ihrem Vorkommen gegenseitig aus, so wird die beobachtete Wahrscheinlichkeit viel kleiner als die zu erwartende Wahrscheinlichkeit. In diesem Fall wäre $I(x, y) << 0$ (vgl. [CH90]). Die Signifikanz der *Kookkurrenz* zwischen zwei *Wortformen* kann anhand der Häufigkeiten des gemeinsamen und einzelnen Auftretens wie folgt berechnet werden ([HQW06],S.213)):

$$sig_{MI}(t_i, t_j) = log_2 \frac{k * k_{ij}}{k_i * k_j} \tag{3}$$

k : Korpusgröße
k_{ij} : Anzahl der Fenster, in denen die Wortformen t_i und t_j enthalten sind
k_i : Anzahl der Fenster, welche die erste Wortform enthalten
k_j : Anzahl der Fenster, welche die zweite Wortform enthalten

Die Korpusgröße dient der Normalisierung der Signifikanzwerte, um berechnete Signifikanzwerte, die aus unterschiedlichen Textkorpora stammen, vergleichen zu können. Der Nachteil der Mutual Information ist, dass die Korrelation seltener Wortpaare im Korpus stark überschätzt wird ([Dun93]).

Log-Likelihood Ratio Test

Der Log-Likelihood Ratio Test ist im Vergleich zur Mutual Information bezüglich seltenen Ereignissen deutlich robuster, da er nicht dazu tendiert seltenen Wortpaaren ein "Übermaß an" Signifikanz zu unterstellen. Formel (4) beschreibt die Berechnung des Log-Likelihhod-Signifikanzmaßes anhand der Häufigkeiten des gemeinsamen und des einzelnen Auftretens der *Wortformen* t_i und t_j (vgl.[Dun93], [HQW06] S.213).

$$\begin{aligned} sig_{LLR}(t_i, t_j) = k * log_2(k) - k_i * log_2(k_i) - k_j * log_2(k_j) \\ + k_{ij} * log_2(k_{ij}) + (k - k_i - k_j + k_{ij}) * log_2(k - k_i - k_j + k_{ij}) \\ + (k_i - k_{ij}) * log_2(k_i - k_{ij}) + (k_j - k_{ij}) * log_2(k_j - k_{ij}) \\ - (k - k_i) * log_2(k - k_i) - (k - k_j) * log_2(k - k_j) \end{aligned} \quad (4)$$

k : Korpusgröße
k_{ij} : Anzahl der Fenster, in denen die Wortformen t_i und t_j enthalten sind
k_i : Anzahl der Fenster, welche die erste Wortform enthalten
k_j : Anzahl der Fenster, welche die zweite Wortform enthalten

2.2.2 Web als Korpus

Das Web ist riesig, kostenlos und für die breite Öffentlichkeit zugänglich. Es ist ständigen Änderungen unterworfen und wird für verbreitete und anerkannte Webseiten regelmäßig aktualisiert. Daher ist das Web für die Nutzung als großer *Textkorpus* für je eine konkrete Lokalisierung sehr interessant. Der Zugriff auf das Web für die automatisierte Verarbeitung wird durch Suchmaschinen Schnittstellen ermöglicht. Es können nach Iryna Gurevych zwei Arten der Nutzung des Webs als *Textkorpus* unterschieden werden (vgl. [Gur]):

1. Zugriff auf statistische Informationen (z.B Trefferanzahl) über die APIs der Suchmaschinen
2. Erstellen von aufgabenspezifischer *Textkorpora* mit Hilfe der Suchmaschinenschnittstellen.

Im ersten Fall sind die Ergebnisse auf der Grundlage des Webs unzuverlässig und oft nicht reproduzierbar, da der Suchmaschinenindex und das Web starken Schwankungen unterworfen sind. Die Interpretation der Ergebnisse wird dadurch erschwert, dass die genaue Zusammensetzung des *Textkorpus* nicht bekannt ist. Bei der zweiten Vorgehensweise können aufgrund der konstanten Datengrundlage zuverlässige und reproduzierbare Ergebnisse erstellt werden. Allerdings ist diese Variante mit einem enormen Aufwand verbunden, da die Daten zur Erstellung eines *Textkorpus* aufbereitet werden müssen. Die extrahierten Texte liegen meist nicht in Reinform vor, sondern sind mit html-Codefragmenten und anderen unbrauchbaren Inhalten (z.B. Werbung) versehen. (vgl. [Gur])

2.3 Eingesetzte Entwicklungswerkzeuge

Die folgenden Abschnitte stellen die verwendeten Werkzeuge MAVEN und JPPF vor. Das Kapitel 2.3.1 beschreibt das Build- und Konfigurationsmanagement-Tool MAVEN und die Maven-Architektur. Das Java Parallel Processing Framework (JPPF) wird anschließend in Kapitel 2.3.2 dargestellt.

2.3.1 MAVEN

Maven ist ein Build- und Konfigurationsmanagement-Tool, das von der Apache Software Foundation entwickelt wurde, um Java-Projekte standardisiert erstellen und verwalten zu können. Maven ist in der Lage für verschiedene Projekte den Build-Prozess zu vereinheitlichen und den Implementierungsaufwand für die Build-Prozesse zu reduzieren. Zur Umsetzung eines Build Prozesses wird ein modellbasierter, deklarativer Ansatz verwendet. Dies bedeutet, dass lediglich der Inhalt des Projektes beschrieben wird und nicht die Struktur oder die Abläufe des Build-Prozesses. Anstatt einem Skript wird ein Projektmodell (POM - Project Object Model) erstellt, welche die Abhängigkeiten, die Projektumgebung und die Projektbeziehungen eines Projekts beschreibt. Diese Metadaten werden in einer strukturierten und einem Schema unterworfenen XML-Datei der `pom.xml` gespeichert. Anhand dieser Daten wird der Build-Prozess automatisch von Maven ausgewertet und ausgeführt. Dies ist ein großer Vorteil gegenüber anderen Build-Tools, wie zum Beispiel Apache Ant, bei denen die Build-Skripte von Hand erstellt und konfiguriert werden müssen. In Maven geht man von immer wiederkehrenden Abläufen in Projekten aus, die in sogenannte Lifecycles (festgelegte Abfolgen von Arbeitsschritten, sogenannten Phasen) abgebildet werden. So bildet Maven den Build- Prozess in einem Standardzyklus ab, der aus einer Abfolge von Phasen besteht. Der Build-Lifecyle enthält zum Beispiel die Phasen `compile, package` und `install`, die ein Projekt kompilieren, in ein Archiv packen und ins lokales Verzeichnis (lokales Repository) installieren. Plugnis können mit den einzelnen Phasen verbunden werden, so dass diese mit der jeweils verknüpften Phase ausgefürt werden. Durch die Verwendung standardisierter Verzeichnisstrukturen und die projektübergreifende identische Benennung und Semantik der Phasen, wird die Einarbeitungszeit in neue Projekt erheblich erleichtert und verkürzt. Einer der wesentlichen Vorteile von Maven ist die Verwaltung von externen Bibliotheken und Komponenten. Benötigte externe Bibliotheken oder Komponenten werden nach dem Festlegen im Projektmodell selbstständig zentral ins lokale Repository aus online zugänglichen Maven-Repositories für alle Projekte kopiert und akualisiert. Transitive Abhängigkeiten, die dabei entstehen können, werden selbstständig analysiert und aufgelöst. Maven löst durch die Repositorys das Problem der Versionsverteilung von neuen Bibliotheken oder Komponenten. Maven unterstützt im größeren Umfang die Erstellung einer Projekt-Homepage, was die Transparenz innerhalb des Projektes verbessert (vgl. [Pop06], [Spi09], [Mas08]).

Maven-Architektur

Maven besteht aus einem Kern und einer beliebigen Anzahl von Plugins. Der Kern interpretiert das Projektmodell und legt die Build-Phasen zur Erstellung des Projekts fest. Alle Build-Phasen zusammen ergeben den Build-Prozess des Projekts. Die einzelnen Phasen werden über Build-Ziele, sogenannte `Goals` ausgeführt. Maven-Plugins fassen eine Reihe von Zielen mit inhaltlich ähnlichen Aufgaben zusammen. Für alle Aufgaben verwendet Maven sogenannte Plugins, die beim ersten Aufruf aus dem online Plugin-Repository geladen und in das lokale Repository installiert werden müssen. Werden externe Bibliotheken im Projekt verwendet, sucht Maven zunächst im lokalen Repository. Werden diese lokal nicht gefunden, wird die Suche auf die Remote-Repositories, die im POM beschrieben werden ausgedehnt und nachinstalliert. Die Auslieferungsdatei, die nach den durchlaufenen Build-Phasen ensteht, kann anderen Projekten im lokalen oder im Remote-Repository zur Verfügung gestellt werden. Somit kann Maven dafür verwendet werden, aus Subsystemen ein komplexes größeres System zu bauen. Die wichtigsten Komponenten von Maven sind in Abbildung 2 dargestellt (vgl. [Pop06]).

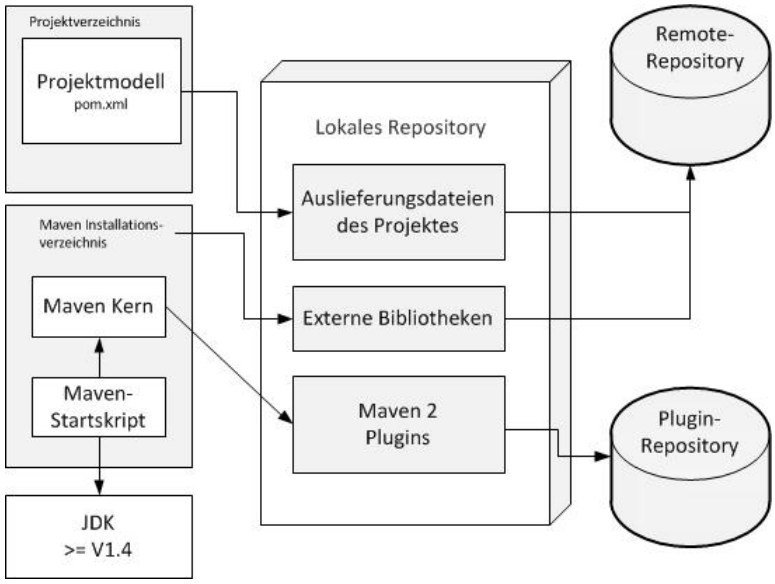

Abbildung 2: Die wichtigsten Architektur-Komponenten von Maven (nach [Pop06], S.80)

2.3.2 JPPF

Die Grid-Computing-Plattform JPPF (Java Parallel Processing Framework) ist ein Open Source Framework, das für die Verteilung rechenintensiver Aufgaben auf unterschiedlichen Rechnern, um die Bearbeitungszeit zu verkürzen, eingesetzt werden kann. JPPF basiert im Wesentlichen auf zwei Aspekten:

1. Aufteilen einer Anwendung in kleinere Teile, die unabhängig und parallel ausgeführt werden können. Das Ergebnis können ein oder mehrere Aufgaben sogenannte `Jobs` sein, welche kleine unabhängige Teilaufgaben, sogenannte `Tasks` beinhalten. Hierbei unterstützt JPPF den Entwickler, um die Aufteilung zu vereinfachen und zu beschleunigen.
2. Ausführen der Anwendung auf einem sogenannten JPPF Grid. Ein JPPF Grid besteht aus einem Server der mit einer beliebigen Anzahl von Knoten kommuniziert. Ein Knoten ist eine JPPF Software-Komponente, die auf einem separaten Rechner installiert und ausgeführt wird. Ein JPPF-Grid stellt somit eine Master/Slave-Architektur dar, bei der die Aufgaben vom Server (Master) auf die Knoten (Slave) verteilt werden.

JPPF Architektur

Ein JPPF Grid besteht aus drei Komponenten, die miteinander kommunizieren:

1. `Client:` Einstiegspunkt in das Netz, die es dem Entwickler mit Hilfe des Client-API ermöglicht die Arbeitspakete (Jobs bzw. Tasks) zu verschicken.
2. `Server:` Erhält die Arbeitspakete eines Clients und verteilt diese zur Bearbeitung auf die Knoten des JPPF-Grids. Sammelt die abgearbeiteten Arbeitspakte der Knoten und schickt diese, nach Erhalt aller Arbeitspakete, wieder an den Client zurück.
3. `Node:` Bearbeitet konkrete Aufgaben (Tasks), die vom Server zugeteilt werden

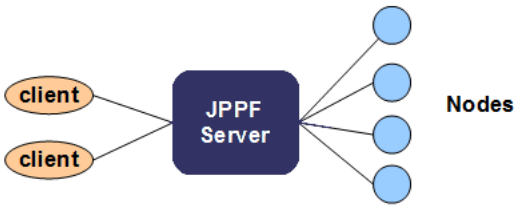

Abbildung 3: JPPF-Grid mit nur einem einzigen JPPF-Server (vgl.[Mat11])

In Abbildung 3 wird der gebräuchlichste Aufbau eines JPPF-Grids gezeigt. Die Clients sind mit einem einzigen Server verbunden, welcher die Aufgaben auf die Knoten verteilt, die Ergebnisse der Knoten sammelt und diese anschließend an die jeweiligen Clients zurücksendet. Ein Nachteil dieser Topologie ist, dass hierdurch ein sogenannter Single Point of Failure (SPOF) entsteht. Fällt der JPPF-Server aus, so führt dies zum gesammten Systemausfall. Um dieses Risiko zu mindern, bietet JPPF die Möglichkeit mehrere Server miteinander in einem Peer-to-Peer-Netzwerk zu verbinden (siehe Abbildung 4).

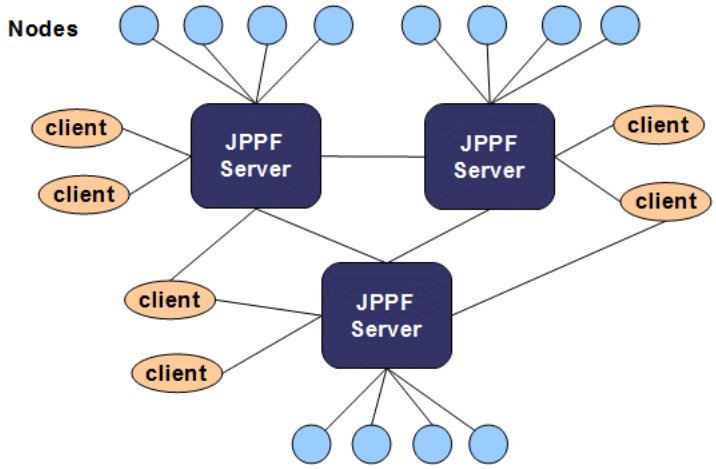

Abbildung 4: JPPF Peer-to-Peer-Netz (vgl. [Mat11])

In Abbildung 4 sind einige Clients mit mehreren Servern verbunden. Fällt ein Server aus, so können sich diese Clients mit anderen Servern verbinden. Dieser `Failover`-Mechanismus der Clients verhindert den Ausfall des gesamten Systems für den jeweiligen Client. Die Knoten besitzen ebenfalls diesen `Failover`-Mechanismus, welcher ihnen ermöglicht, bei einem Serverausfall mit anderen Servern zu kommunizieren. Ein JPPF Peer-to-Peer Netz, wie in Abbildung 4 gezeigt, ermöglicht die Lastverteilung auf dem gesamten Netz. Die Verbindung zwischen zwei Servern ist bidirektional: Besteht eine Verbindung von Server A zu Server B, dann behandelt Server A den Server B als Client und der Server A wird vom Server B als Knoten angesehen. Zu beachten ist, dass in diesem Szenario jeder Server auf lokaler Ebene immer noch als Master fungiert(vgl. [Mat11]).

2.4 API basierte Nutzung von Suchmaschinen

Dieser Abschnitt stellt die Search-APIs der Suchmaschinenbetreiber Google (80% Marktanteil, Stand 2012), Bing (4,6%) und Yahoo (3,4%) vor (vgl. [Web11]). Alle drei Suchmaschinenschnittstellen sind Representational State Transfer (REST)-APIs, was bedeutet, dass eine Abfrage mit einem einfachen http-request oder innerhalb eines Webbrowsers mittels einer URL und verschiedenen Parametern an die Suchmaschine geschickt werden können, um die jeweils benötigten Informationen in einem kompakten Datenformat (JSON,XML etc.) zu erhalten.

BOOS (Build your Own Search Service) von Yahoo
BOOS ermöglicht es Entwicklern, das Web oder bestimmte Webseiten nach Inhalten zu durchsuchen und diese mit eigenen Daten zu vermischen. Suchergebnisse können mittels des Search-API erzeugt und als JSON, XML oder JSON-P zurückgegeben werden. Das API ermöglicht den vollen Zugriff auf den *Index*, die Crawlertechnik und auf das Ranking. Dadurch sind Entwickler in der Lage, die Schnittstelle zu nutzen, um eigene Suchprodukte zu bauen. Das Search-API ist seit Ende 2011 kostenpflichtig und wird je nach Abfragetyp (Full Web, Image Only etc.) in unterschiedliche Preiskategorien und Limitierungen gestaffelt. Beispielsweise kosten 1000 Suchanfragen $ 0.80 (Stand Jan 2012, vgl. [yah12]).

JSON/Atom Custom Search API von Google
Das JSON/Atom Custom Search API ermöglicht die direkte Abfrage von Suchergebnissen in JSON oder Atom-Format. Die Nutzung ist für bis zu 100 Abfragen pro Tag kostenlos. Alle weiteren 1000 Abfragen kosten $ 5 und sind auf ein Maximum von 10.000 Abfragen pro Tag limitiert. Ein größeres Abfragelimit kann bei Google beantragt werden (Stand Jan 2012, vgl. [goo12]).

BING API, Version 2
Das Bing API ermöglicht es, eine flexible und leistungsfähige Suchmaschine für Suchanfragen zu nutzen. Die Suchergebnisse können in XML oder JSON-Format zurückgeliefert werden und die Abfragedaten können auf einen bestimmten Quelltyp (z.B. Web oder News) eingegrenzt werden. Die Nutzung mit Hilfe einer AppId, die man bei Bing beantragen kann, ist kostenlos. Dieser Umstand macht das Bing API für den Einsatz im Rahmen von Projekten im Forschungsumfeld besonders interessant, da somit keine überhöhten Kosten für experimentelle Umgebungen anfallen (vgl. [bin12]).

3 Entwurf und Implementierung

Die Signifikanz von *Kookkurrenzen* kann mittels statistischen Signifikanzmaßen auf der Grundlage von Worthäufigkeiten in *Textkorpora* berechnet werden (vgl. Abschnitt 2.2). Das Java-API `co-weight`, das im Rahmen dieser Arbeit entwickelt wurde, greift hierfür unter zur Hilfenahme einer Suchmaschinenschnittstelle auf Textinhalte aus dem Web zu. Dieser Zugriff ermöglicht es, die benötigten Häufigkeitswerte zur Berechnung der Signifikanz von Kookkurrenzen zu ermitteln, welche später als Gewichte der Kanten (Konzeptrelationen) im Wiki-Gaphen verwendet werden sollen. Im nächsten Kapitel wird der Entwurf und die Implementierung des `co-weight` APIs sowie der entwickelten Werkzeuge beschrieben.

3.1 Performance Anforderungen an das co-weight API

Die Berechnung aller Signifikanzwerte des deutschprachigen Wiki-Graphen benötigt 1.669.722 *Abfragen* und für den englischsprachigen Wiki-Graphen 8.750.703 *Abfragen*. Die Bearbeitungszeit würde bei einer sequentiellen *Abfrage* des Suchmaschinenindex, unter der Annahme einer Abfragedauer von 1,5 ms pro Anfrage, ca. 28 Tage in Anspruch nehmen. Die des englischsprachigen Graphen würde ca. 5 Monate beanspruchen. Um die zeiteffiziente Berechenbarkeit der Kantengewichte zu gewährleisten, werden mit Hilfe des JPPF-Frameworks, die *Abfragen* parallelisiert (siehe Kapitel 2.3.2). Der JPPF-Cluster, welcher für mehrere unterschiedliche wissenschaftliche Experimente aufgebaut wurde, verwendet zum Zeitpunkt der Ausarbeitung 39 inhomogene Dual-Core-Rechner. Davon werden 38 Rechner für die JPPF-Nodes und 1 Rechner für den JPPF-Server (Driver) verwendet. Da die Rechner mit zwei Hauptprozessoren ausgestattet sind, werden die *Abfragen*, welche die Nodes ausführen, mittels des JPPF-Frameworks auf die zwei Rechnerkerne verteilt. Dies bedeutet, dass 76 *Abfragen* bei maximaler Auslastung der Rechner- und Softwarekomponenten parallel an die Suchmaschine geschickt werden können. Die Bearbeitungszeit der *Abfragen* könnte somit theoretisch für den deutschprachigen Graphen auf ca. 6 Stunden und für den englischsprachigen Graphen auf ca. 2 Tage reduziert werden.

3.2 Systementwurf

Das Gesamtsystem besteht aus drei Komponenten, die miteinander interagieren (siehe Abbildung 5):

1. Java-API `co-weight`
2. JPPF-Server
3. JPPF-Nodes

Das co-weight API verarbeitet die Konzeptrelationen (Kanten) zu *Abfragen*. Die *Abfragen* werden auf eine vom Benutzer festgelegte Anzahl von Threads sogenannte JPPF-Tasks verteilt. Diese Tasks werden in einem JPPF-Job gebündelt und an den

JPPF-Server geschickt, der die Tasks auf die einzelnen Nodes verteilt. Die Nodes führen die Tasks aus und schicken somit die *Abfragen* an die zuvor ausgewählte Suchmaschine. Der JPPF-Server verwaltet die Teilergebnisse der abgearbeiteten Tasks und schickt das Gesamtergebnis (Treffer aller Abfragen) an die `co-weight` Komponente zurück. Diese berechnet aus den erhaltenen Ergebnissen die Signifikanzwerte mittels eines vom Benutzer ausgewählten Signifikanzmaßes und erstellt einen gewichteten *Wiki-Graphen*.

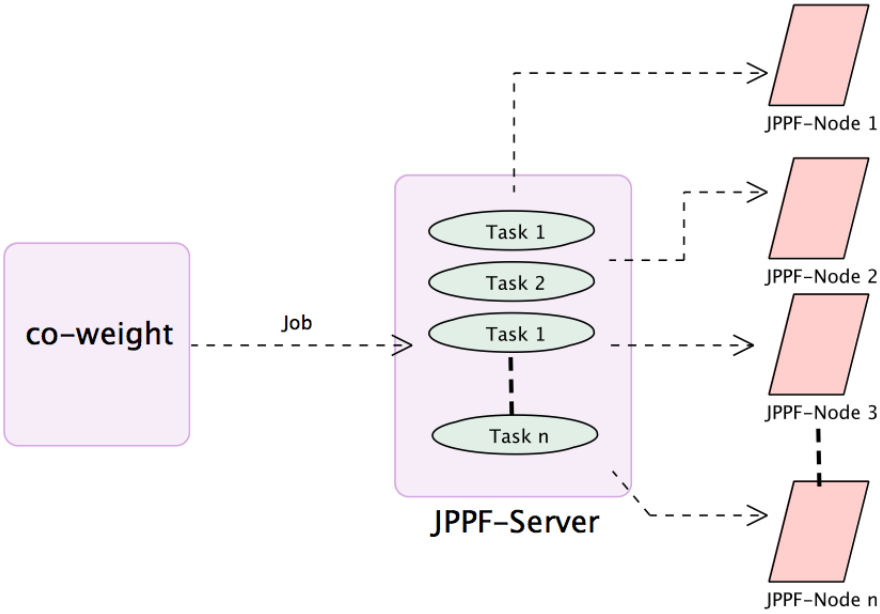

Abbildung 5: Interaktionsdiagramm mit den drei Teilkomponenten: co-weight-Client, dem JPPF-Server und den JPPF-Nodes.

3.2.1 Modulübersicht

Das `co-weight` System besteht aus zwei selbst implementierten und vier externen Modulen (siehe Abbildung 6). Das selbst implementierte Modul `co-weight API` bietet dem Benutzer mehrere Funktionen, um die Gewichtung eines beliebigen *Wiki-Graphen* anhand von Suchmaschinen-Indizes durchführen zu können. Dafür greift es auf die drei externen Module `TULUM, JPPF` und `JSON` zu.

Das **Modul TULUM** enthält das WikiGraph-Framework und die abgespeicherten *Wiki-Graphen* in Form von komprimierten GZIP oder BZ2-Archiven.

Der `WikigraphIOService` und der `WikiGraphClassPathSearchService`, die im Wikigraphen-Framework implementiert sind, werden für das Laden einer *WikiGraph* Instanz im `co-weight API` verwendet.

Das **JPPF** Modul enthält mehrere Werkzeuge, um den benötigten JPPF-Client zu implementieren, der im Kapitel 3.5.2 vorgestellt wird.

Das Modul **JSON** bietet die Möglichkeit, Daten die in JSON konvertiert sind, in JSON-Objekte zu parsen. Diese Funktionalität wird auf die jeweilige erhaltene Response der Suchmaschine angewendet, um die *Suchtreffer* zu ermitteln.

Das Modul **coweight-analyzer** wurde für die Auswertung der *Suchtreffer* und der berechneten Signifikanzwerte implementiert. Der coweight-analyzer verwendet hierfür die Module TULUM und JFreeChart.

Das Modul **JFreeChart** enthält mehrere Methoden um Charts für Auswertungszwecke zu generieren.

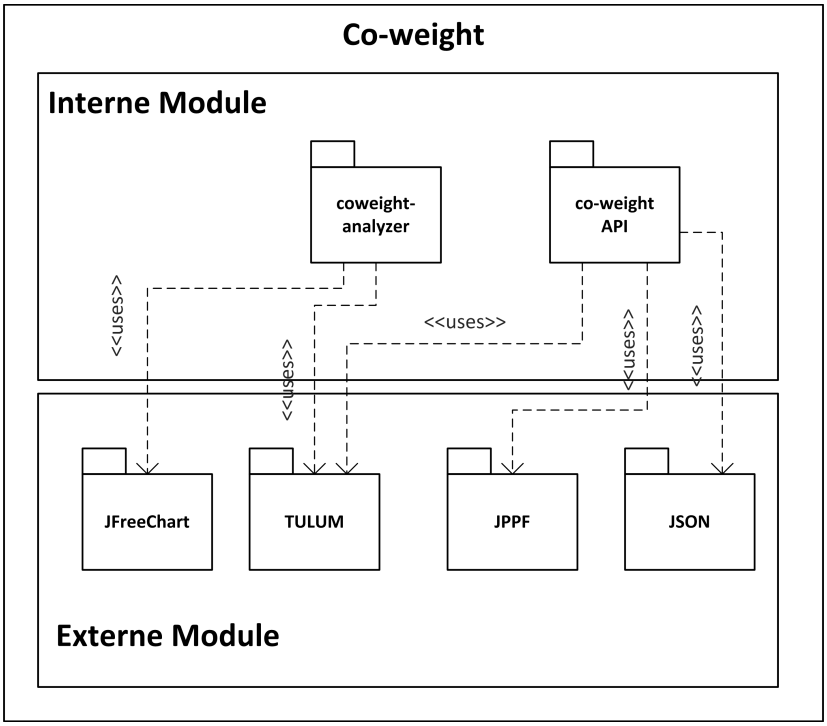

Abbildung 6: Modulübersicht zu co-weight zur Verdeutlichung von Abhängigkeiten zu externen Bibliotheken und Modulen.

3.2.2 Die zentralen Klassen

Im folgenden Abschnitt werden die wichtigsten Klassen und Funktionen des Moduls `co-weight api` vorgestellt (vgl. Abbildung 7).

Eine Instanz der Klasse `CooccurrenceUnit` stellt eine zu gewichtende Kante (Konzeptrelation) des *Wiki-Graphen* dar. Diese enthält die Namen der Knoten, mit der die Kante verbunden ist, die *Suchtreffer* für die jeweiligen *Abfragen* und den entsprechenden Signifikanzwert der *Konzeptrelation*. Drei *Suchtreffer* werden für die Berechnung des Signifikanzwerts benötigt. Ein *Suchtreffer* für den ersten Knoten [a], ein Suchtreffer für den zweiten Knoten [b] und schließlich den Suchtreffer für die kombinierte Konzeptabfrage [a+b].

Die Klasse `CooccurrenceEstimator` implementiert alle notwendigen Funktionen, um die benötigten *Suchtreffer* zu ermitteln, die Signifikanzwerte zu berechnen und den *Wiki-Graphen* zu gewichten. Auf die verschiedenen Konfigurationsmöglichkeiten wird in Kapitel 3.5.1 eingegangen.
Der `CooccurrenceEstimator` verwendet den `WikiGraphClassPathSearchService` und den `WikiGraphIOService` aus dem Modul TULUM, um eine *Wiki-Graph* Instanz zu laden, welche die benötigten Konzeptrelationen enthält. Diese Konzeptrelationen werden für die weitere Verarbeitung auf eine Sammlung von `CooccurenceUnit` Instanzen abgebildet.

Die Klasse `JPPFService` realisiert den JPPF-Client, der für die Kommunikationen mit dem JPPF-Server und das Erstellen und Versenden der JPPF Jobs verantwortlich ist (vgl. Abschnitt 2.3.2).

Die Klassen `JPPFQueryListTask` und `JPPFTripleQueryListTask` können als die kleinsten atomaren Arbeitspakete im `co-weight` System angesehen werden. Beide Klassen implementieren das `Interface Runnable`, welches sie von der Klasse `JPPFTask` erben. Der Unterschied zwischen den beiden Klassen liegt in der unterschiedlichen Datenverarbeitung. Eine `JPPFQueryListTask` Instanz verarbeitet eine Sammlung von *Termen*, wohingegen eine `JPPFTripleQueryListTask` Instanz eine Sammlung von `CooccurencUnit` Instanzen verarbeitet. Diese zwei Varianten werden für die Realisierung der unterschiedlichen Suchstrategien benötigt, welche im Kapitel 3.3 vorgestellt werden

Der `JPPFResultListener` implementiert das `TaskResultListener` Interface aus dem JPPF Modul. Diese Klasse ist für die Überwachung und das Verarbeiten der einzelnen Task-Ergebnisse des JPPF Servers zuständig.

Die Schnittstellen `SearchStrategy` und `CalculationStrategy` geben Implementierungsrestriktionen bzgl. der Signifikanzmaße und Suchanfragen vor.

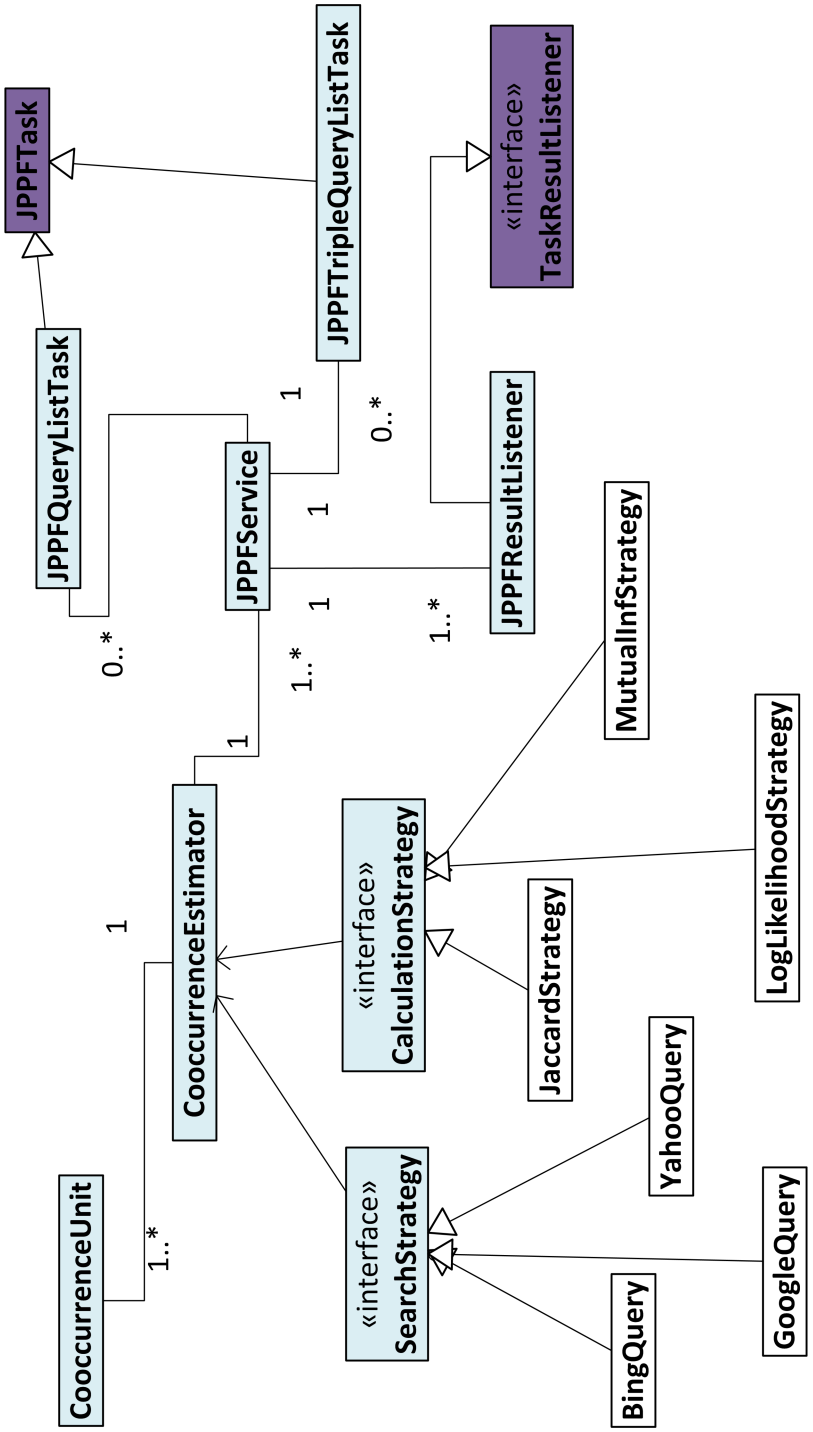

Abbildung 7: Die zentralen Klassen von `co-weight` und ihre Beziehungen zueinander

3.2.3 Vereinfachter Prozessablauf

In Abbildung 8 werden die Teilprozesse und der Prozessablauf des `co-weight` Systems veranschaulicht, die im folgenden Abschnitt erläutert werden.

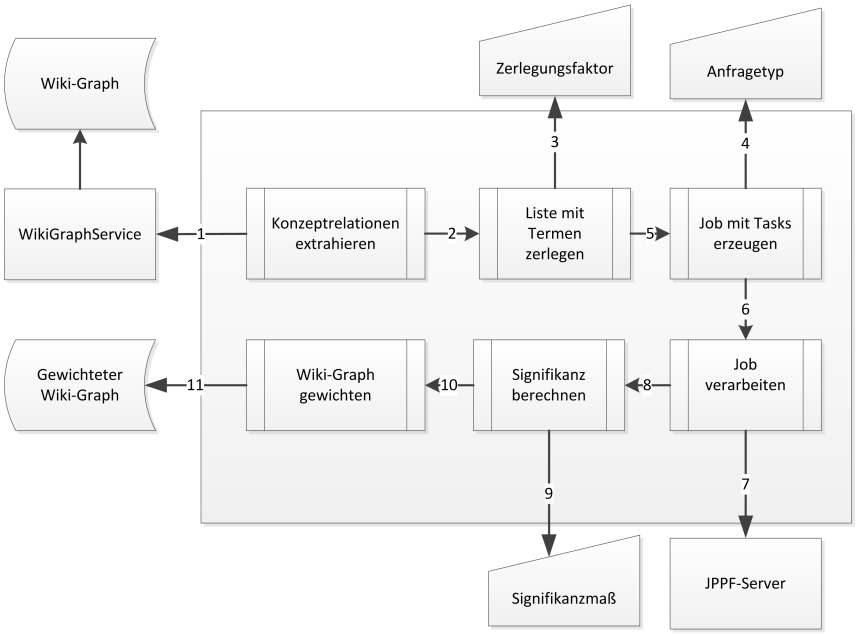

Abbildung 8: Vereinfachter Prozessablauf des `co-weight` Systems

Zu Beginn der Prozesskette muss festgelegt werden, welche Sprache des *Wiki-Graphen* geladen werden soll. Zusätzlich muss die Größe der Arbeitspakete für jeden `JPPF-Task` bestimmt werden, welche Suchmaschine und welches Signifikanzmaß angewendet werden soll.
Durch den Aufruf des `WikiGraphClassPathSearchService` und der vom Benutzer vorgegebenen Sprache, wird der sprachspezifische *Wiki-Graph*, der vom `co-weight` System aus dem Module TULUM referenziert wird, in den Hauptspeicher geladen. Aus der geladenen *Wiki-Graph* Instanz werden, mittels einer Tiefensuche, die Kanten extrahiert und auf `CooccurrenceUnit` Instanzen abgebildet. Diese Sammlung an `CooccurrenceUnit` Instanzen wird anhand der zuvor festgelegten Taskgröße (Zerlegungsfaktor) in kleiner Teillisten zerlegt, die anschließend die Arbeitspakete der JPPF-Nodes bilden.
Zu beachten ist, dass der Zerlegungsfaktor nicht zu groß gewählt werden darf, da die `JPPF-Nodes`, welche die Tasks bearbeiten einen begrenzten Arbeitsspeicher zu Verfügung haben. Für jedes Arbeitspaket ist eine `JPPF-Task` Instanz verantwortlich, welche im `JPPFService` erzeugt wird. Die Task-Instanzen werden mittels einer `JPPF-Job` Instanz an den `JPPF-Server` geschickt, welcher die Tasks auf die `JPPF-Nodes` verteilt.

Bei der Ausführung der Tasks auf den `JPPF-Nodes` werden die einzelnen Arbeitspakte abgearbeitet. Die Tasks erzeugen aus den `CooccurrenceUnit`-Instanzen die *Suchanfrage-Terme*, welche in einem HTTP-Request an die vom Benutzer ausgewählte Suchmaschine geschickt werden. Aus der erhaltenen HTTP-Response werden die *Suchtreffer* extrahiert und in die `CooccurrenceUnit` Instanzen zur späteren Weiterverarbeitung gesetzt.

Im Codeauschnitt 1 wird das Erzeugen, Versenden und Verarbeiten einer *Abfrage* an die Suchmaschine BING demonstriert. Die dort verwendete Abfrage-URL besteht aus suchmaschinenspezifischen Parametern (Adresse, Land, Datenübertragungformat etc) und dem *Term*. Als Response wird ein String in JSON-Format erwartet, der mit Hilfe des JSON Parsers aus dem JSON Module verarbeitet wird.
Nach Bearbeitung aller Arbeitspakete wird der Job mit allen `CooccurrenceUnit` Instanzen, die nun jetzt die *Suchtreffer* enthalten an den `JPPFService` zurück gesendet. Die `CooccurrenceUnit` Instanzen gehen anschließend in den ausgesuchten Signifikanzberechnungsalgorithmus ein, der die Berechnung der Signifikanzwerte der einzelnen *Konzeptrelationen* anhand der ermittelten *Suchtreffer* durchführt. Im letzten Schritt wird der *Wiki-Graph* mit den Signifikanzwerten gewichtet und mit Hilfe des WikiGraphen-Frameworks abgespeichert.

```
1  public long submitQuery(String queryTerm)
2                              throws SearchEngineQueryException {
3
4    BufferedReader reader= null;
5      try { queryTerm = URIUtil.encodeQuery(queryTerm, UTF_8);
6
7      URL url = new URL("http://api.bing.net/json.aspx?AppId="+
8            APPLICATION_ID+"&Version=2.2"+MARKET+&Query=+queryTerm+
9                "&Sources=web+spell&Web.Count=1&JsonType=raw");
10
11     URLConnection connection = url.openConnection();
12
13     String line = null;
14     StringBuilder builder = new StringBuilder();
15
16          reader = new BufferedReader(new
17             InputStreamReader(connection.getInputStream()));
18
19         while((line = reader.readLine()) != null){
20           builder.append(line);
21         }
22
23     String response = builder.toString();
24     JSONObject jObject = new JSONObject(response);
25
26     return(Long.parseLong(jObject.getJSONObject(SEARCH_RESPONSE).
           getJSONObject(WEB).getString(TOTAL))); ...
```

Listing 1: Der Codeauschnitt zeigt das Erzeugen Versenden und Verarbeiten einer Abfrage an die Suchmaschine BING

3.3 Suchstrategien

In diesem Kapitel werden die zwei Strategien zur Ermittlung der *Suchtreffer* vorgestellt, die dem Anwender im `co-weight` System zur Verfügung stehen

3.3.1 Triple-Strategy

Die `Triple-Strategy` bildet die *Konzeptrelationen* eins zu eins auf die `CooccurrenceUnit` Instanzen, aus denen die *Terme* für die *Abfragen* generiert werden, ab. Eine *Konzeptrelation* repräsentiert die Kante (Beziehung) zwischen zwei Knoten (Konzepten) im *Wiki-Graphen*. Die Knotennamen [a,b] werden für die Generierung der *Terme* [a,b, a+b] verwendet. Die `Triple-Strategy` erzeugt diese *Terme* für jede *Konzeptrelation* neu. Dies führt zu redundanten *Abfragen*, da ein Knoten (Konzept) in verschieden *Konzeptrelationen* auftauchen kann. Zum Beispiel wird für die Konzeptrelationen (`Knochen, Osteoporose`) und (`Knochen, Knochenmark`) zwei mal die *Abfrage* Knochen an die Suchmaschine geschickt. Der Vorteil bei dieser Strategie ist, dass einzelne *Konzeptrelationen* unabhängig voneinander gewichtet werden können. So können beispielweise bei einer Erweiterung eines Graphen die Gewichte nur für die Graphenerweiterung berechnet werden. In Abbildung 9 wird die `Triple-Strategy` veranschaulicht.

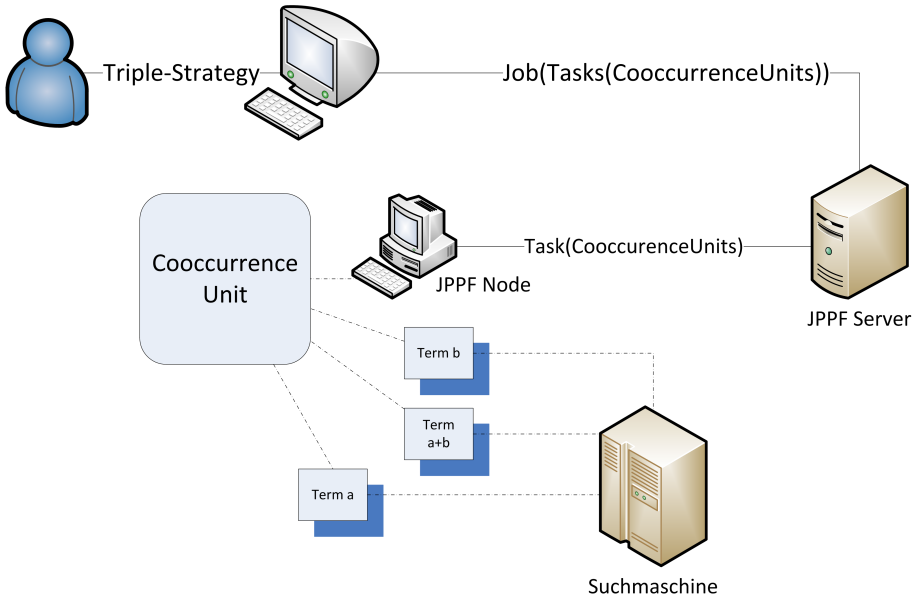

Abbildung 9: Ablaufdiagramm der Triple-Strategy des co-weight Systems

3.3.2 OneDouble-Strategy

Die `OneDouble-Strategy` wurde entwickelt, um redundante *Abfragen* zu vermeiden, die es bei der im vorigen Abschnitt vorgestellten `Triple-Strategy` gibt. Die `OneDouble-Strategy` ermittelt zunächst die *Suchtreffer* der Konzepte [*Term* a, *Term* b] des gesamten *Wiki-Graphen* und überträgt diese auf `CooccurrenceUnit` Instanzen (*Konzeptrelationen*). Anschließend werden die *Suchtreffer* für die *Terme* [a+b] in einem separaten, d.h. durch einen neuen Suchanfragedurchlauf ermittelt. Diese Vorgehensweise reduziert die Abfrageanzahl um ca. ein Drittel.

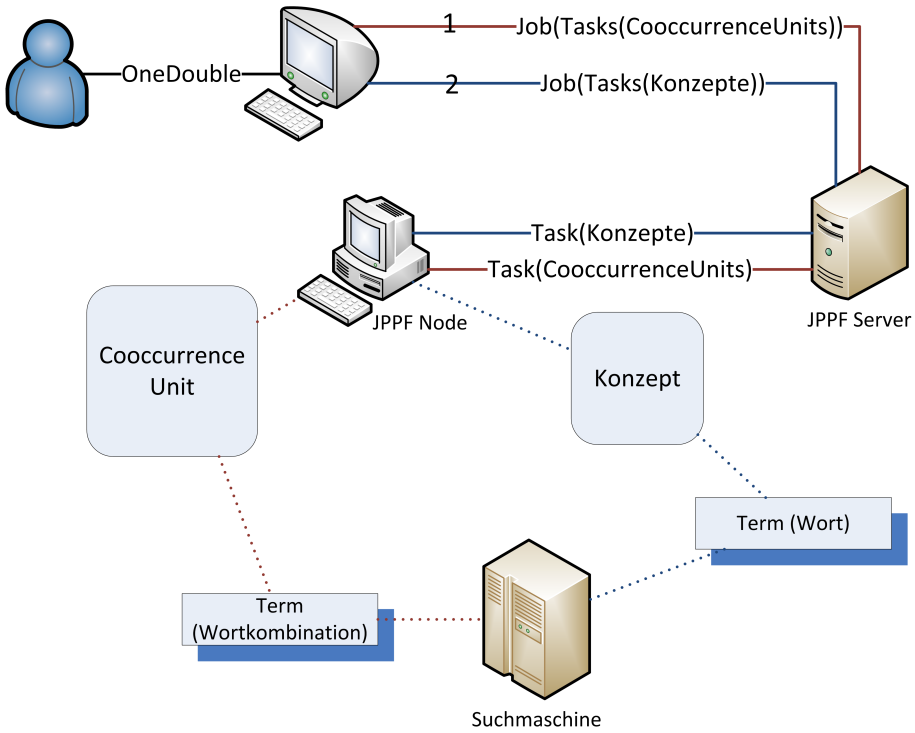

Abbildung 10: Ablaufdiagramm der OneDouble-Strategy des co-weight Systems

Der Unterschied zur `Triple-Strategy` besteht darin, dass die *Abfragen* nicht auf einen einzigen JPPF-Job verteilt werden, sondern auf zwei. Es werden zuerst alle *Terme* der *Konzeptrelation* über das JPPF-Grid an die Suchmaschine geschickt und danach ein zweiter, der die kombinierten *Abfragen* [Term a+b] der *Konzeptrelationen* enthält.

3.4 Die Systemfunktionalität von co-weight

In diesem Abschnitt werden die Benutzerfunktionalität und die Resume-Funktionalität des co-weight Systems vorgestellt.

3.4.1 Die Benutzerfunktionalität

Das co-weight System bietet dem Benutzer eine Reihe von Konfigurationsmöglichkeiten an:

1. Die Auswahl zwischen den Suchstrategien `Triple` und `OneDouble`
2. Die Wahl zwischen den Signifikanzmaßen `Jaccard`, `Log-Likelihood` und `Mutual Information`
3. Die Sprache, in der der *Wiki-Graph* gewichtet werden soll
4. Die Wahl zwischen den Suchmaschinen `Google`, `Bing` und `Yahoo`
5. Das Festlegen der Größe der Arbeitspakete für den einzelnen Task

Der Benutzer kann den gesamten Prozessablauf, oder aber nur Teilprozesse konfigurieren und ablaufen lassen. Hierfür bietet das `co-weight` System die Möglichkeit die *Konzeptrelationen* nach den *Abfragen* mit den *Suchtreffern* in eine CSV-Datei zu speichern, diese wieder einzulesen und zu bearbeiten. So kann die Berechnung der Signifikanzwerte und die Gewichtung des *Wiki-Graphen* zu einem späteren Zeitpunkt erfolgen.

3.4.2 Die Resume-Funktionalität

Das JPPF-Framework ermöglicht es dem Entwickler sehr schnell und relativ einfach ein JPPF-Grid aufzubauen. Da diese Open-Source-Software sich noch in der Entwicklungsphase befindet, sind einige Funktionen noch nicht ausgereift. So kann es zum Beispiel mit der verwendeten JPPF-Version 2.4 passieren, dass ein JPPF-Node der noch nicht mit seinen Tasks fertig ist, abstürzt und der JPPF-Server dies nicht registriert. Die Folge hiervon ist, dass der JPPF-Server auf die noch ausstehenden Tasks des ausgefallenen JPPF-Nodes wartet und die Ergebnisse zurück hält. Dieser Zustand kann nur mit einem Neustart des JPPF-Servers aufgehoben werden, was zu einem Verlust der bereits ermittelten Ergebnisse führen würde. Aufgrund dieser Problematik bietet der `co-weight-service` eine Resume-Fuktionalität an. Diese stellt eine Art Fallback-Mechanismus dar, welcher bei unerwartetem Systemausfall die Wiederherstellung des Systemzustands vor dem Absturz übernimmt.

Die Realisierung der Resume-Funktionalität

Der `JPPFService` des `co-weight` APIs der für das Erzeugen und Versenden der Jobs verantwortlich ist, sichert nach dem Versenden des Jobs, die initialisierten Tasks in einer Datei. Der `JPPFResultListener` überwacht den JPPF-Server und speichert in Hunderterschritten die abgearbeiteten Tasks in einer zweiten Datei ab. Kommt es zum Systemausfall, dann lädt das System die Benutzerkonfigurationen, die vor dem Start separat gespeichert wurden ins System, vergleicht die gespeicherten initalisierten Tasks mit den gespeicherten Ergebnissen des Listeners und ermittelt die noch offen stehenden Tasks. Anschließend werden die noch offenen Aufgabenpakete in einem Job an den JPPF-Server geschickt und weiter verarbeitet.

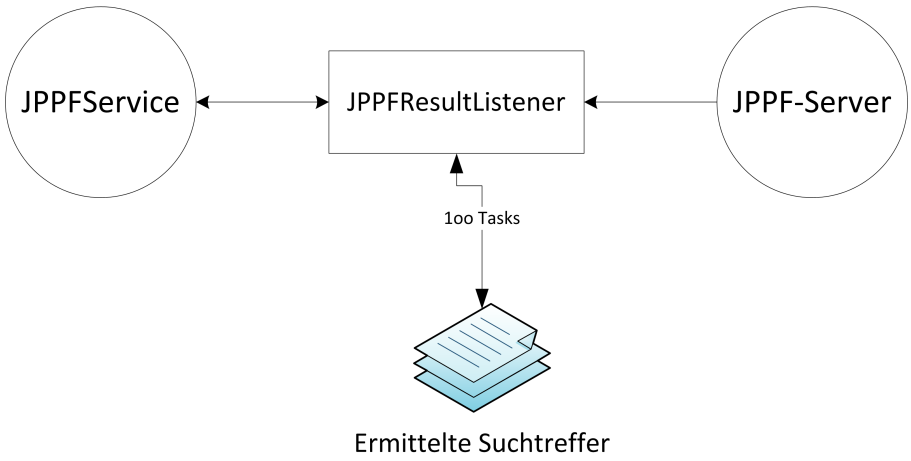

Abbildung 11: Der `JPPFResultListener` als Fallback-Meachanismus. Die abgearbeiten Tasks werden vom Listener abgeholt und in Hunderterschritten in einer seperaten Datei gespeichert.

3.5 Implementierungsdetails

In diesem Kapitel wird die Realisierung der Erweiterbarkeit des `Cooccurrence Estimator` und der `JPPFService`, der den JPPF-Client implementiert vorgestellt.

3.5.1 CooccurrenceEstimator

Der `CooccurrenceEstimator` bietet Methoden an, um die vorgestellte Benuzerfunktionalität zu realisieren. Die `computeSignificance` Methode in Listing 2 ist für das Berechnen der Signifikanzwerte im `CooccurenceEstimator` verantwortlich. Diese ruft zur Laufzeit die polymorphe Methode `computeSignificance` auf, welche in Abhängigkeit vom Objekttyp, den entsprechenden Signifikanzberechnungsalgorithmus durchführt. Hierfür wurde ein `Interface` erstellt, welche die abstrakte Methode `computeSignificance` mit einem Parameter für die Korpusgröße und einem Paramter für eine Liste von `CooccurrenceUnit` Instanzen definiert (siehe Listing 3). Dieses `Interface` wird für die Implementierung des jeweiligen Signifikanzberechnungsalgorithmus verwendet. Zur Verdeutlichung ist die Implementierung des Jaccard Signifikanzmaß in Listing 4 dargestellt. Der Vorteil bei diesem Implementierungsentwurf ist, dass sich das `co-weight` API ohne großen Aufwand um eine neue Signifikanzberechnungsmethode erweitern lässt. Dazu muss nur eine neue Klasse erstellt werden, welche die `computeSignificance` Methode des Interfaces spezifisch durch den Berechnungsalgorithmus implementiert. Auf die gleiche Weise sind die unterschiedlichen Suchmaschinenanfragen implementiert, um Änderungen am `JPPFService` zu vermeiden.

```
1 //Use a reference to a Strategy object
2 public class CooccurrenceEstimator {
3
4   public void computeSignificance(CalculationStrategy strategy) {
5
6           strategy.computeSignificance(new BigInteger(CORPUS_SIZE),
              cooccurrenceUnits);
7     }
8 }
```

Listing 2: Methode welche die Signifikanzwerte im CooccurenceEstimator berechnet. Der Berechnungsalgorithmus hängt vom übergebenen Objekt-Typ ab

```
1
2 // The context class uses this to call the concrete strategy.
3 public interface CalculationStrategy {
4
5     public abstract void computeSignificance(BigInteger corpus,
6       List<CooccurrenceUnit> coUnits);
7 }
```

Listing 3: Interface das die polymorphe Methode zur Berechnung der Signifikanzwerte definiert

```
// Implements the compute algorithm using the strategy interface
public class Jaccard2Strategy implements CalculationStrategy {

  @Override
  public void computeSignificance(BigInteger corpus,
      List<CooccurrenceUnit> coUnits) {

    for(CooccurrenceUnit u : coUnits){

       double nab = new Long(u.getCombinedHits()).doubleValue();
       double na = new Long( u.getFirstConceptHits()).doubleValue();
       double nb = new Long(u.getSecondConceptHits()).doubleValue();

       double sig = nab / (na+nb-nab);
       u.setSignificance(sig);
    }
  }
```

Listing 4: Methode die die Berechnung der Signifikanzwerte mitteles des Jaccard Signifikanzmaß implementiert

3.5.2 JPPF-Service

Der `JPPFService` implementiert den `JPPF-Client`, der zwei Arten von Jobs generieren kann. Einen Job, der `JPPFTripleQueryListTask` Instanzen enthält, die eine Liste von `CooccurenceUnit` Instanzen bearbeitet. Der andere Job beinhaltet `JPPFQueryListTask` Instanzen, die eine Liste von Konzepten (Knotennamen) verarbeiten (vgl. Kapitel 3.3). Für beide Varianten wurden jeweils drei Methoden implementiert, die für die Verbindung mit dem `JPPF-Server`, das Erzeugen und Versenden des `JPPF-Jobs` und das Extrahieren der Ergebnisse aus den bearbeiteten Tasks verantwortlich sind. In den nachfolgenden Codeauschnitten wird die Implementierung am Beispiel der `TripleStrategy` im Detail dargestellt.

```
public List<CooccurrenceUnit> createJPPFCoUnitConnection (List<List<
    CooccurrenceUnit>> lists, SearchStrategy searchStrategy){

 jppfClient = new JPPFClient();
 JPPFJob job = null;
 List<CooccurrenceUnit> totalCoResults = new ArrayList<
                                          CooccurrenceUnit>();
  try {  job = createCoUnitsJob(lists, searchStrategy);
         totalCoResults.addAll(executeCoUnitsJob(job));

    } catch (Exception e) {
         logger.error(e.getLocalizedMessage(),e);
    }finally{
         if(jppfClient != null) {
                  jppfClient.close();
      }
    }return totalCoResults;
  }
```

Listing 5: Erzeugen einer Verbindung mit dem JPPF-Server

Die `createJPPFCoUnitConnection` Methode erzeugt eine Verbindung zum `JPPF-Server` durch das Erstellen eines neuen `JPPF-Client`-Objekts. Durch den Aufruf des Konstruktors des `JPPF-Clients` (Listing 5, Zeile 6) wird die konfigurierte Properties-Datei des `JPPF-Clients` geladen, welche die TCP/IP Einstellungen für die Verbindung mit dem `JPPF-Server` beinhaltet. Anschließend generiert sie mit Hilfe der `createCoUnitsJob`-Methode und den übergebenen `CooccurrenceUnit` Instanzen einen `JPPF-Job`, führt diesen mit der `executeCoUnitJob`-Methode aus und gibt die `CooccurrenceUnit` Instanzen mit den ermittelten *Suchtreffern* zurück.

```
1  private JPPFJob createCoUnitsJob(List<List<CooccurrenceUnit>>
       searchLists,SearchStrategy searchStrategy) throws JPPFException{
2
3      JPPFJob  job = new JPPFJob();
4      jobID="TULUM-job-"+System.currentTimeMillis();
5      job.setId(jobID);
6
7      for(int i=0; i < searchLists.size(); i++){
8
9        JPPFTask task = new JPPFTripleQueryListTask(searchLists.get(i),
             searchStrategy);
10       task.setId("unit"+i);
11       job.addTask(task);
12     }
13
14     return job;
15
16   }
```

Listing 6: Erzeugen eines Jobs mit Tasks die CooccurrenceUnits enthalten

Die `createCoUnitsJob` Methode (Listing 6) erzeugt einen Job mit Tasks. Jedem Tasks wird eine Liste mit `CooccurrenceUnit` Instanzen als Arbeitspaket übergeben. Die `JPPFTripleQueryListTasks` erzeugen aus den `CooccurenceUnit` Instanzen die benötigten *Terme* und ermitteln mit der entsprechenden Suchmaschine, die durch das übergebene `SearchStrategy` Objekt festgelegt wird, die *Suchtreffer* auf dem entsprechenden JPPF-Node.

```
1  private List<CooccurrenceUnit> executeCoUnitsJob(JPPFJob job) throws
       ExecuteJPPFException {
2
3      ....
4
5    job.setBlocking(false);
6    JPPFResultListener resultCollector = new JPPFResultListener(job.
       getTasks().size());
7    job.setResultListener(resultCollector);
8
9    jppfClient.submit(job);
10   logger.info("<<job submission was successful>>");
11
12   while(!resultCollector.isJobComplete()){
13        Thread.sleep(1000);
14   }
15
16   tasks = resultCollector.getResults();
17
18   for(JPPFTask task : tasks){
19      if(task instanceof JPPFTripleQueryListTask){
20
21       JPPFTripleQueryListTask tripleTask = (JPPFTripleQueryListTask)
           task;
22
23       searchResults = (List<CooccurrenceUnit>) tripleTask.getResult();
24       totalCoResults.addAll(searchResults);
25
26       }
27       ......
28
29       return totalCoResults;
30   }
```

Listing 7: In dieser Methode wird der erzeugte Job versendet und die anschließend erhaltenen Ergebnisse aus den abgearbeiteten Tasks extrahiert

In Listing 7 wird das Versenden und anschließende Verarbeiten des Jobs gezeigt. `JPPF` bietet zwei Varianten des Verschickens der `JPPF-Jobs` an. Die eine ist der sogenannte Block-Modus, bei dem das Programm solange blockiert bleibt, bis es den bearbeiteten Job vom `JPPF-Server` zurück bekommt. Die andere Variante, der Non-Block-Modus, wird mit einem selbst implementierten Listener realisiert, welcher die Überwachung der Tasks übernimmt. Der `JPPFResultListener`, welcher in Zeile 6 erzeugt wird, holt einen Task sobald dieser von einem JPPF-Node bearbeitet wurde vom `JPPF-Server` ab und sammelt so anstatt des `JPPF-Servers` die Ergebnisse. Dieser Listener informiert den `JPPF-Client`, sobald alle Tasks abgearbeitet wurden. Dies wird durch das Vergleichen der Anzahl erhaltener mit der Anzahl der initialisierten Tasks realisiert. Sobald alle Tasks abgearbeitet sind, kann die Sammlung an Tasks abgerufen werden. Die einzelnen `CooccurrenceUnit` Instanzen, die mit den Tasks verknüpft sind, werden in einer `Collection` aggregiert und zur weiteren Verarbeitung weitergereicht.

4 Ergebnisse & Diskussion

Dieses Kapitel ist in zwei Teile unterteilt. Im ersten Abschnitt werden die Ergebnisse der *Suchtreffer* der Suchmaschine BING analysiert. Diese sind für die Berechnung der Signifikanzwerte erforderlich. Der zweite Teil beschäftigt sich mit den Ergebnissen der verschiedenen Signifikanzmaße für die Gewichtung der Wiki-Graphen.

4.1 Bing-Suchanfragenauswertung

Eine *Konzeptrelation* besteht aus zwei Knoten a,b und einer Kante c. Um eine *Konzeptrelation* zu gewichten, benötigt man drei Suchtreffer. Der Suchtreffer für den ersten *Term* [a] (Knoten), den Suchtreffer für den zweiten *Term* [b] (von Knoten b über Kante c) und schließlich den Suchtreffer für die kombinierte Konzeptabfrage [a+b] (siehe Kapitel 2.2). Der deutschsprachige *Wiki-Graph* besitzt 556.574 *Konzeptrelationen* und der englischsprachige *Wiki-Graph* 2.916.901 *Konzeptrelationen*. Dies bedeutet, dass für den deutschsprachigen Wiki-Graphen 1.669.722 Suchanfragen und für den englischsprachigen Wiki-Graphen 8.750.703 Suchanfragen benötigt werden. Die einzige Suchmaschine die dafür in Frage kommt ist BING. Da BING als alleinige Suchmaschine kostenfrei Suchanfragen ermöglicht und einen hinreichend großen *Such-Index* besitzt.

Die Ergebnisse der Suchanfragen wurden hinsichtlich der Verwertbarkeit der *Suchtreffer* zur Berechnung von *Signifikanzwerten* untersucht. Dabei wurden die *Konzeptrelationen* in zwei Klassen aufgeteilt:

a) Brauchbare Konzeptrelationen

b) Unbrauchbare Konzeptrelationen

Alle abgefragten *Konzeptrelationen* (a, b, a+b) die keinen *Suchtreffer* für einen ihrer drei *Terme* erzielt haben, wurden der Klasse der unbrauchbaren Konzeptrelationen zugeordnet und das `co-weight` System wies diesen *Konzeptrelationen* einen Signifikanzwert von 0 zu. Diese Klasse wird in 7 Gruppen von A bis G unterteilt. In Gruppe A werden alle *Konzeptrelationen* gesammelt, die in allen drei *Termen* keinen *Suchtreffer* erzielt hatten. Die Gruppe B enthält alle *Konzeptrelationen* die für die *Terme* a und b keine *Suchtreffer* erzielten. In der Gruppe C und D sind die *Konzeptrelationen* die für den *Term* a oder b und den *Term* a+b keinen *Suchtreffer* aufwiesen. In den Gruppen E, F und G sind die *Konzeptrelationen* die nur einen *Term* mit keinem *Suchtreffer* enthielten, nach dem jeweiligen *Term* aufgeteilt.

Der deutschsprachige Wiki-Graph[1] (d.h. alle enthaltenen *Konzeptrelationen*) wies dabei eine Verlustrate von 18 % auf (siehe Abbildung 12). Beim englischsprachigen Wiki-Graph[2] waren die Verluste mit 20% etwas höher (siehe Abbildung 13).

[1] healthgraph-german june-2011
[2] healthgraph-english june-2011

Bing-Suchtreffer für den deutschsprachigen Wiki-Graphen [21.10.2011]

Graph	Brauchbare Konzeptrelationen	Unbrauchbare Konzeptrelationen	Gesamtanzahl
healthgraph-german june-2011	458079	98495	556574

Gruppen	A	B	C	D	E	F	G
Unbrauchbare Konzeptrel.	a=0, b=0, a+b=0	a=0, b=0	a=0, a+b=0	b=0, a+b=0	a=0	b=0	a+b=0
98495	3459	5	21656	29469	123	402	43381

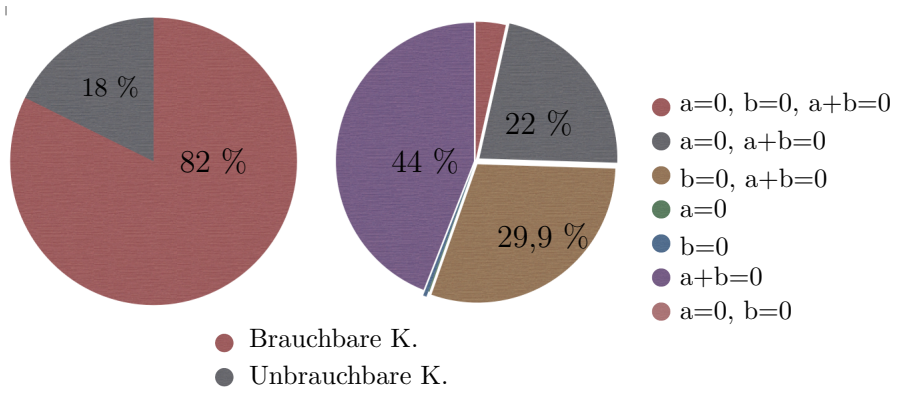

Abbildung 12: Anteilige Analyse der Konzeptrelationen für den Wiki-Graph "healthgraph-german june-2011", ungefiltert, BING.

Bing-Suchtreffer für den englischsprachigen Wiki-Graphen [08.11.2011]

Graph	Brauchbare Konzeptrelationen	Unbrauchbare Konzeptrelationen	Gesamtanzahl
healthgraph-english june-2011	2165598	751303	2916901

Gruppen	A	B	C	D	E	F	G
Unbrauchbare Konzeptrel.	a=0, b=0, a+b=0	a=0, b=0	a=0, a+b=0	b=0, a+b=0	a=0	b=0	a+b=0
751303	2151	2	27162	34800	292	388	686508

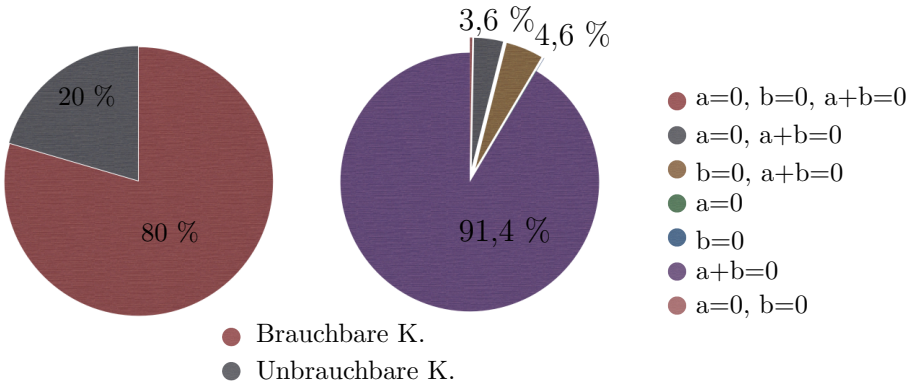

Abbildung 13: Anteilige Analyse der Konzeptrelationen für den Wiki-Graph "healthgraph-english june-2011", ungefiltert, BING.

4.1.1 Ungefilterte Abfrage des BING-Indexes

Nach einer genaueren Analyse der *Terme* stellte sich heraus, dass gelegentlich Klammerausdrücke und Unterstriche als Ersatz für Leerzeichen in den *Termen* vorhanden waren `(z.B.Abstrich (Medizin), Kapillare (Anatomie), Kommunikationsprobleme_bei_der_Arztvisite, Manifeste_Atmungsinsuffizienz)`. Um diesen Umstand zu kompensieren werden derartige semantische Ergänzungen davon betroffene *Konzeptrelation* durch einen pattern-basierten Filter zunächst "gesäubert". Ein weiterer Grund für die beobachte Verlustrate ist, dass BING bei einer Vielzahl von parallelen *Abfragen* das Abfragezeitfenster kurzzeitig schließt. Dies führte zu Fehlermeldungen, welche sich in den *Konzeptrelationen* in 0 Werten äußert. Daher muss vorher die "optimale"Anzahl an parallel laufenenden Abfragen ermittelt werden. Nach

Berücksichtigung dieser Einschränkungen, lag die Verlustrate für den deutschsprachigen Wiki-Graphen nur noch bei 1 % (siehe Abbildung 14). Für den englischsprachigen Wiki-Graphen reduzierte sich die Verlustrate von zunächst 20 % ebenfalls auf lediglich 1% (siehe Abbildung 15).

Bing-Suchtreffer für den deutschsprachigen Wiki-Graphen [29.10.2011]

Graph	Brauchbare Konzeptrelationen	Unbrauchbare Konzeptrelationen	Gesamtanzahl
healthgraph-german june-2011	549104	7470	556574

Gruppen	A	B	C	D	E	F	G
Unbrauchbare Konzeptrel.	a=0, b=0, a+b=0	a=0, b=0	a=0, a+b=0	b=0, a+b=0	a=0	b=0	a+b=0
7470	98	0	989	1733	16	105	4529

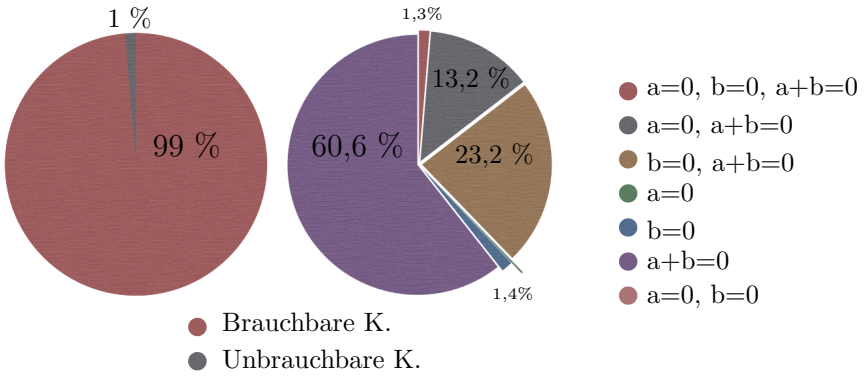

Abbildung 14: Anteilige Analyse der Konzeptrelationen für den Wiki-Graph "healthgraph-german june-2011", gefiltert, BING.

Bing-Suchtreffer für den englischsprachigen Wiki-Graphen [18.11.2011]

Graph	Brauchbare Konzeptrelationen	Unbrauchbare Konzeptrelationen	Gesamtanzahl
healthgraph-english june-2011	2901844	15057	2916901

Gruppen	A	B	C	D	E	F	G
Unbrauchbare Konzeptrel.	a=0, b=0, a+b=0	a=0, b=0	a=0, a+b=0	b=0, a+b=0	a=0	b=0	a+b=0
15057	54	0	3236	4066	8	32	7661

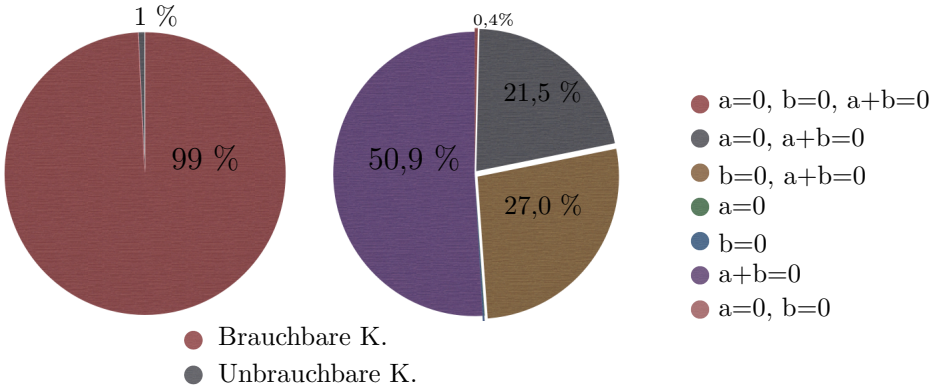

Abbildung 15: Anteilige Analyse der Konzeptrelationen für den Wiki-Graph "healthgraph-english june-2011", gefiltert, BING.

4.1.2 Gefilterte Abfrage des BING-Indexes

Die 7.470 unbrauchbaren *Konzeptrelationen* des deutschsprachigen Wiki-Graphen bestehen aus 5.449 verschiedenen *Termen*, die nicht gefunden werden konnten (Die Kombination aus [a+b] wird hier als ein eigenständiger *Term* im Sinne einer Suchanfrage gewertet). Unter den 98 *Konzeptrelationen* für die Gruppe A (siehe Abbildung 14) existieren insgesamt 53 verschiedene *Terme*, die alle einen sexuellen Kontext hatten (z.B. `Penisvergrößerung, Analfisting, Tiersex` etc.). In der Gruppe C befinden sich 34 verschiedene *Terme*, von denen sind 24 mit einem sexuellen Kontext behaftet. Diese Abfrageterme werden von BING aus Jugendschutzgründen mit Hilfe einer sogenannten Block-List geblockt. Bei den *Konzeptrelationen* in Gruppe D handelt es sich um 819 verschiedene *Terme*, von denen

die meisten Eigennamen zu real existierenden Personen waren (z.B. `Dr. Rieper, John F. Fulton, Christl R. Vonholdt etc.`). Die Gruppe F mit 12 verschiedenen *Termen* und die Gruppe G mit 4.529 verschiedenen *Termen* wiesen keine Besonderheiten mehr auf. Es handelte sich hierbei in den Gruppen D,F und G um *Terme* die zum Zeitpunkt der Analyse im *Index* von BING nicht vorhanden bzw. abrufbar waren. In der Gruppe E sind es nur 2 verschiedene *Terme* (`-itis und -zele`), die ein unmittellbar vor dem Wort vorangestelltes Minuszeichen besitzen. Dieses Minuszeichen führt aufgrund der Suchsyntax von BING zum Ausschluß aus der Suche.

4.2 Beurteilung der Signifikanzwerte zur Abbildung der semantischen Nähe

Dieser Abschnitt befasst sich mit der Validierung der berechneten Signifikanzwerte, welche die semantische Nähe von medizinischen Konzepten beschreiben sollen. Mit Hilfe der Suchmaschine BING wurde das Zählen der Kookkurrenzen und der Häufigkeit der einzelnen *Wortformen* im Web realisiert, welche zur Berechnung der Signifikanzwerte herangezogen wurden. Durch eine Evaluation soll geprüft werden, ob die berechneten Signifikanzwerte eine sinnvolle und nachvollziehbare semantische Nähe abbilden. Um eine solche Evaluation durchzuführen, wurde nach einem Gütekriterum gesucht, das die Annahme bestätigt, dass die berechneten Signifikanzwerte ein gutes Maß für die semantische Nähe der Konzepte sind. Die Weiterleitungen innerhalb Wikipedias eignen sich näherungsweise als ein solches Gütekriterium. Sie deuten auf einen engen semantischen Zusammenhang zwischen zwei Konzepten hin, da Weiterleitungen aufgrund von Synonymen, Sammelartikeln oder unterschiedlicher phonetischer Schreibweisen entstehen. Derartige Weiterleitungen werden innerhalb des *Wiki-Graphen* durch einen bestimmten Kantentyp[1] abgebildet. Die berechneten Signifikanzwerte dieses Kantentyps sollen die enge semantische Nähe ausdrücken und können deshalb als Gütekriterum für die Abbildung der semantischen Nähe verwendet werden. Dies bedeutet, dass für Weiterleitungen sehr hohe Signifikanzwerte im Vergleich zu anderen Konzepten zu erwarten sind. Die Berechnung der Signifikanzwerte wurde für die Signifikanzmaße Jaccard, Mutual Information und Log-Likelihood durchgeführt. In den Berechnungsalgorithmus der Signifikanzmaße Mutual Information und Log-Likelihood fließt die *Textkorpusgröße* mit ein. Da in dieser Arbeit der *Textkorpus* Web als Textsammlung dient und der Zugriff auf das Web über die Suchmaschine BING erfolgt, wurde die Größe der indizierten Webseiten der Suchmaschine BING als *Textkorpusgröße* verwendet. Die Schätzung der Anzahl indizierter Webseiten wurde aus der Webseite `worldwidewebsize.com` entnommen, auf der die Größe des Webs mittels eines Verfahrens, das im Rahmen einer Masterarbeit an der Universität Tilburg entstanden ist, geschätzt wird (siehe [dK11]).

[1] REDIRECT

In den folgenden Abbildungen werden die Häufigkeitsverteilungen für die Signifikanzwerte aller Kantentypen im Verhältnis zu der Häufigkeitsverteilung der Signifikanzwerte der Weiterleitungen für alle berechneten Signifikanzmaße dargestellt. Ein niedriger Signifikanzwert drückt hierbei eine schwache Zusammengehörigkeit der Konzepte und ein hoher Signifikanzwert eine starke Zusammengehörigkeit der Konzepte aus.

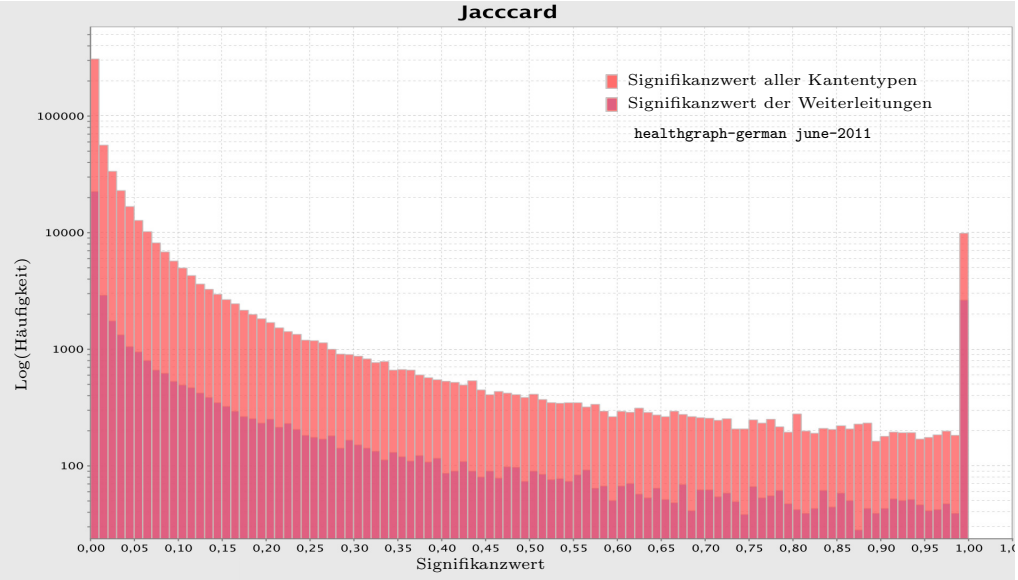

Abbildung 16: Häufigkeitsverteilung der berechneten Signifikanzwert nach Jaccard für den deutschsprachigen Graphen (Skalen nicht normiert, in Rohform belassen)

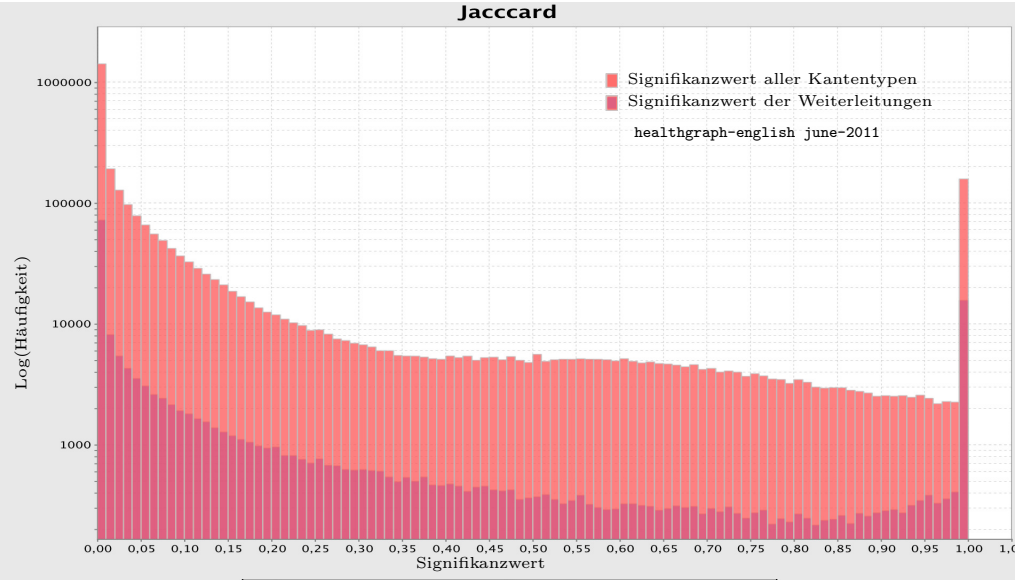

Abbildung 17: Häufigkeitsverteilung der berechneten Signifikanzwert nach Jaccard für den englischsprachigen Graphen (Skalen nicht normiert, in Rohform belassen)

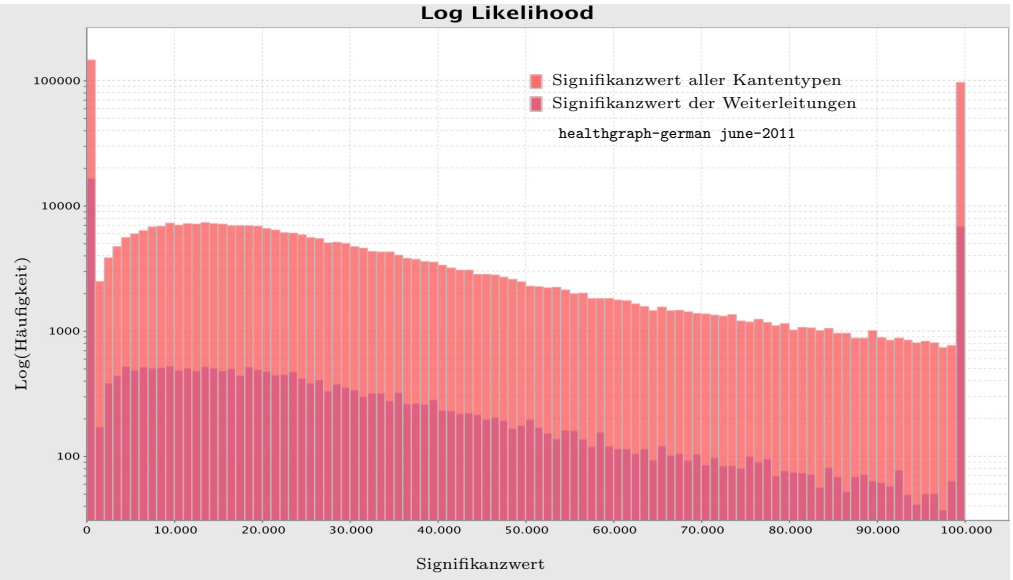

Abbildung 18: Häufigkeitsverteilung der berechneten Signifikanzwert nach Log Likelihood für den deutschsprachigen Graphen (Skalen nicht normiert, in Rohform belassen)

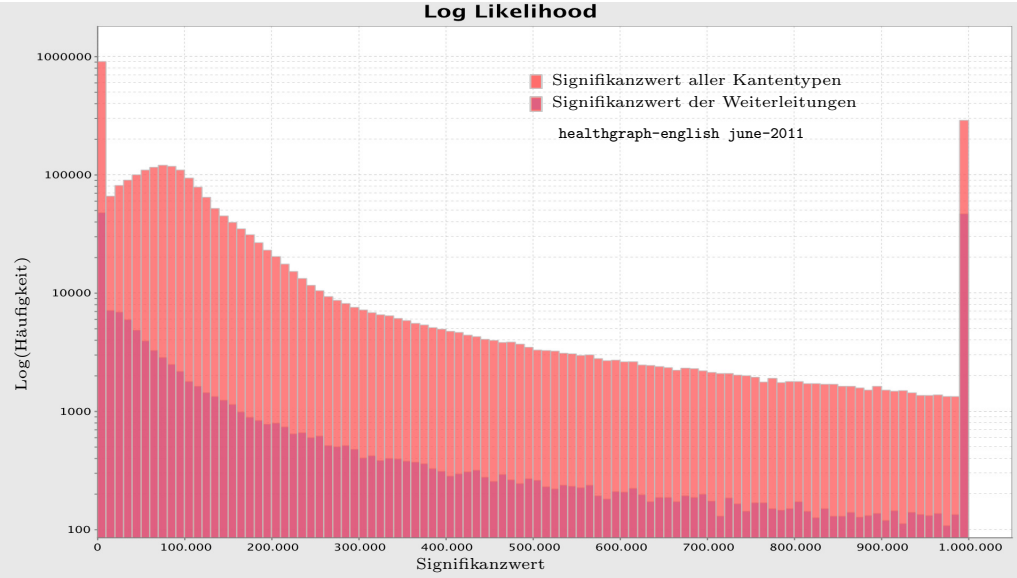

Abbildung 19: Häufigkeitsverteilung der berechneten Signifikanzwert nach Log Likelihood für den englischsprachigen Graphen (Skalen nicht normiert, in Rohform belassen)

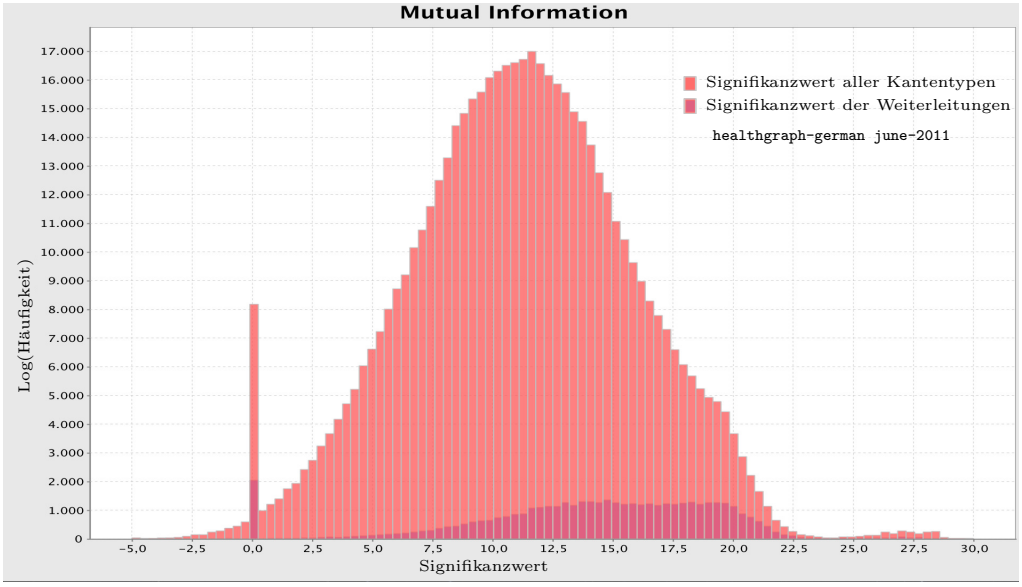

Abbildung 20: Häufigkeitsverteilung der berechneten Signifikanzwert nach Mutual Information für den deutschsprachigen Graphen (Skalen nicht normiert, in Rohform belassen)

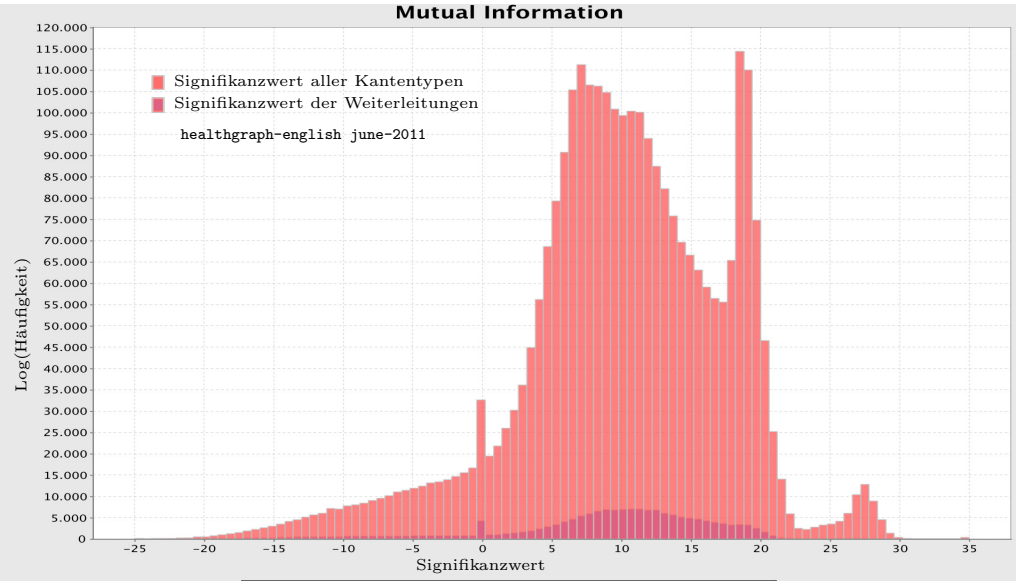

Abbildung 21: Häufigkeitsverteilung der berechneten Signifikanzwert nach Mutual Information für den englischsprachigen Graph (Skalen nicht normiert, in Rohform belassen)

Die logarithmische Darstellung der Signifikanzwerte nach dem Jaccard-Maß, welche wegen der unterschiedlichen Größenordungen gewählt wurde, zeigt einen deutlichen exponentiellen Abfall der Signifikanzwerte der Weiterleitungen und der Signifikanzwerte aller Knotentypen an (siehe Abb. 16 und Abb. 17). Der Grund für diesen exponentiellen Abfall ist, dass die überwiegende Anzahl der *Konzeptrelationen* einen sehr kleinen Wert für die kombinierte Konzeptabfrage im Verhältnis zu ihren einzelnen Konzeptwerten enthielten[1]. Der erwartete Unterschied, dass die Verteilung der Häufigkeiten der Weiterleitungen überwiegend im Bereich der hohen Signifikanzwerte liegen müssten, ist für den Jaccard nicht eingetreten. Die logarithmische Darstellung der Häufigkeiten der Signifikanzwerte des Log-Likelihood Maßes zeigen ebenfalls, dass die Verteilung der Signifikanzwerte der Weiterleitungen und die Verteilung der Signifikanzwerte aller Kantentypen sich nicht unterscheiden (siehe Abb. 18 und Abb. 19). Die Abbildungen der Häufigkeitsverteilungen der Mutual Information (siehe Abb. 20 und Abb. 21) zeigen eine leichte Verschiebung der Häufigkeiten der Weiterleitungen im Vergleich zu allen Knotentypen. Dies liegt daran, dass die Korrelation seltener Wortpaare von der Mutual Information stark überbewertet wird und ist somit eher kein Indiz für die Stärke der semantischen Nähe von Weiterleitungen. Die Diagramme widerlegen die Annahme, dass die mittels *Suchtreffer* der Suchmaschine BING berechneten Signifikanzwerte, die semantischen Nähe der Konzepte abbilden.

[1]Beispiel: Blutausstrich: 2830 Cytauxzoonose: 1690 Kombiniert: 4

Gründe für die Häufigkeitsverteilung der Signifikanzwerte

a) Eine häufig auftretende Konzeptrelation ist der Kantentyp `Kategorie auf Artikel`. Meist handelt es sich dabei um einen allgemeinen Oberbegriff, der auf einen medizinischen Fachbegriff verweist. Die Suchtreffer für den allgemeinen Oberbegriff waren in der Regel relativ hoch. Deutlich weniger Suchtreffer wurden für den spezifischen Fachbegriff erzielt. Die für die Berechnung der Signifikanz nötige kombinierte Abfrage von Ober- und Fachbegriff lieferte die geringsten Ergebnisse. Im allgemeinen Sprachgebrauch bekannte medizinische Oberbegriffe wurden also deutlich häufiger im Web gefunden als spezifische medizinische Fachterme. Dieser Umstand führte dazu, dass eine geringe Signifikanz bzgl. der Koookkurenz und somit eine geringe semantische Nähe errechnet wurde. [2]

b) Ein weiterer Grund ist, dass der Index der Suchmaschine Bing länderspezifischen Einschränkungen unterworfen ist. Wird eine Abfrage an die Suchmaschine geschickt, so enthält diese einen sogenannten `Market`. Ein `Market` ist ein Parameter der Abfrage, der angibt, in welcher Sprache gesucht werden soll. Da es selbst für den deutschsprachigen Wiki-Graphen vorkommen kann, dass kombinierte Konzeptabfragen auch englischsprachige Konzepte enthalten, führte eine solche länderspezifische Abfrage zu einer vergleichsweise sehr geringen Suchtrefferanzahl.

[2] Beispiel: Bindegewebe:312000 Alaria alata:8710 Kombiniert:15

5 Fazit und Ausblick

Das abschließende Kapitel fasst die Ergebnisse zusammen und bewertet selbige gegenüber den gesetzten Zielen.

5.1 Bewertung

Das Ziel, eine Softwarekomponente zur automatisierten Berechnung von Konzeptrelations-Signifikanzwerten zu implementieren, konnte erreicht werden. Das entwickelte `co-weight` System emöglicht die automatisierte Berechnung von Signifikanzwerten. Es implementiert drei verschiedene Berechnungsalgorithmen und ist so konzipiert, dass eine Erweiterung leicht realisiert werden kann. Das System ist plattformunabhängig in Java implementiert und verteilt in einem Cluster beliebige Jobs an mehrere Rechenknoten.
Der Verteilungsalgorithmus, welcher mit der Hilfe des JPPF-Frameworks realisiert wurde, ermöglicht unter idealen Bedingungen (d.h. idealer Auslastung der zur Verfügung stehenden Cluster-Knoten und Suchmaschinen ohne Anfrage-Limitierungen) eine zeiteffiziente Berechnung von Kantengewichten. Selbst große Mengen an Konzeptrelationen[1] können ohne Schwierigkeiten in einer angemessen Zeit verarbeitet werden. Das JPPF-Grid, welches für die Verteilung der Arbeitspakete eingesetzt wird, sorgt für eine zeiteffiziente Berechnung. Im Idealfall, d.h. bei maximaler Auslastung der Software und Hardwarekomponenten, konnte die Berechnung innerhalb einer Stunde für den deutschsprachigen Graphen erfolgen, was bei einer sequentiellen Abfrage ca. 28 Tage gedauert hätte. Die Ausnutzung der theoretisch erreichbaren maximalen Parallelität der Abfragen wurde jedoch im Verlauf der Entwicklung nicht immer erreicht. Dies lag zum Teil daran, das BING die parallele Anfragezahl limitierte, wodurch die Berechnungzeit für den deutsprachigen Graphen bei ca. 12 Stunden und für den englischsprachigen Graphen bei ca. 2 Tagen lag.
Das Ziel ein Suchmaschinen unabhängiges API zur Kookkurrenzberechnung zu implementieren, konnte durch ein Entwurfsmuster realisiert werden, welches dem Benutzer des APIs erlaubt, die Abfrage an eine Suchmaschine seiner Wahl anzupassen. Das Gewichten von Graphen-Instanzen anhand der berechneten Signifikanzwerte wurde durch eine Reihe von implementierten Werkzeugen umgesetzt. Die mit Hilfe der Suchmaschine Bing automatisch berechneten Signifikanzwerte liefern bisher jedoch keine aussagekräftigen Signifikanzwerte (vgl Abschnitt 4.2).

[1] WikiGraph-EN: zur Zeit ca. 4.8 Millionen Graphkanten

5.2 Optimierungspotential

Das Ziel Graph-Instanzen anhand der berechneten Signifikanzwerte tatsächlich sinnvoll zu gewichten, konnte somit nicht hinreichend zufriedenstellend erreicht werden. Hier besteht Bedarf an Optimierungen bzw., um Vergleichswerte berechnen zu können, der Zugriff auf ein Index von anderen Suchmaschinen über ein sogenanntes Public-API. Leider hat sich im Verlauf der Arbeit herausgestellt, dass zumindest Google und Yahoo, diese Art der programmatischen Benutzung des Public-API kostenpflichtig gemacht haben. Hier könnte durch einen akademischen Lizenz-Key jedoch Abhilfe geschaffen werden. Somit könnten weitere Analysen bestätigen, ob sich der generelle Ansatz, Signifikanzwerte von Textkookkurrenzen über öffentliche Suchmaschinen zu berechnen eignet oder nicht. Eine derartige Validierung des Grundansatzes, könnte anhand von anderen Suchmaschinen-Anbietern bei Vorliegen eines validen volumenunlimitierten Zugriffsschlüssels zeitnah erfolgen.
Die realisierten Signifikanzmaße Log-Liklihood und Mutual Information benötigen zur Berechung der Signifikanzwerte die Textkorpusgröße. In dieser Arbeit wurde hierfür ein Schätzwert der gesamt Gesamtgröße des Suchindex von BING für den deutschsprachigen Graphen und den englischsprachigen Graphen verwendet. Aus diesem Grund sind die Signifikanzwerte für die unterschiedlichen Sprachen (Deutsch, Englisch) nicht vergleichbar. Um diese Signifikanzwerte vergleichbar zu machen, müsste die Größe des deutschen und des englischen Index von BING geschätzt werden.

Die Verbesserung des Abfragealgorithmus, der die kombinierte Nutzung von verschiedenen sogenannten Markets der Suchmaschine BING ermöglicht, um höhere Trefferzahlen zu erzielen, bietet ebenfalls Möglichkeiten zur Optimierung. Somit könnte eine Homogenisierung der Absoluttrefferzahlen im BING-Index resultieren. Es ist hierdurch zu erwarten, dass die bisher geringen kombinierten Trefferzahlen für Begriffe wie 'Pubmed' und 'Medline' in deutschsprachigen WikiGraphen deutlich höher ausfallen sollten. Hierdurch würden sich je nach gewähltem Signifikanzmaß auch bessere Signifikanzwerte ergeben. Derartige Beispiele für in der medizinisch geprägten Fachsprache typischen und aus dem Englischen entlehnten Fachbegriffen finden sich häufig.

Abschließend lässt sich festhalten, dass die Ergebnisse der vorliegenden Arbeit möglicherweise ein Ausgangspunkt für weiterführende Bachelor- oder Masterarbeiten sein können.

6 Referenzen

Literatur

[bin12] bing. Bing api version 2. URL: http://msdn.microsoft.com/en-us/library/dd251056.aspx, [Stand: 24. Januar 2012]. [Last checked: 24. Januar 2012].

[CH90] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography, 1990.

[dK11] Maurice de Kunder. Worldwiedewebsize daily estimated size of the world wide web. URL: http://www.worldwidewebsize.com, [Stand: 28. November 2011]. [Last checked: 28. November 2011].

[Dun93] Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *COMPUTATIONAL LINGUISTICS*, 19(1):61–74, 1993.

[goo12] google. Json/atom custom search api. URL: http://code.google.com/intl/de-DE/apis/customsearch/v1/overview.html, [Stand: 24. Januar 2012]. [Last checked: 24. Januar 2012].

[Gur] Iryna Gurevych. Das world wide web als computerlinguistische ressource.

[Hei11] Hochschule Heilbronn. Tulum. URL: http://www.hs-heilbronn.de/574903/tulum, [Stand: 28. November 2011]. [Last checked: 28. November 2011].

[HQW06] Gerahrd Heyer, Uwe Quasthoff, and Thomas Witting. *Text Mining: Wissensrohstoff Text*. W3L GmbH, 2006.

[Mas08] Van Zyl J. Porter B. Casey J. Sanchez C. Massol, V. Better builds with maven - the how-to guide for maven 2.0. Global:USA, July 2008.

[Mat11] Parallel Matters. Jppf java parallel processing framework. URL: http://www.jppf.org, [Stand: 21. Januar 2011]. Last checked: 21. Januar 2011].

[Pop06] Gunther Popp. *Konfigurationsmanagement mit Subversion, Ant und Maven: Ein Praxishandbuch für Software-Architekten und Entwickler*. dpunkt, Heidelberg, 2006.

[Pre10] Dirk Preuten. Der index der suchmaschinen. URL: http://www.conversionmedia.de/presseveroeffentlichungen/kurznachrichten/der-index-der-suchmaschinen, [Stand: 02. Juni 2010]. [Last checked: 24. Januar 2012].

[Spi09] M. Spiller. *Maven 2: Konfigurationsmanagement mit Java*. mitp bei Redline. mitp-Verlag, 2009.

[Tri10] Benjamin Trinczek. WikiGraph Analyse, Extraktion und Abbildung von medizinischen Strukturmerkmalen der Wikipedia durch Realisierung eines flexiblen Graphen-Frameworks, 2010.

[Web11] WebHits. Nutzung von suchmaschinen. URL: `http://www.webhits.de/deutsch/index.shtml?/deutsch/webstats.html`, [Stand: 21. Januar 2011]. Last checked: 21. Januar 2011].

[Wik11a] Wikipedia. Personal health record. URL: `http://en.wikipedia.org/wiki/Personal_health_record`, [Stand: 11. Oktober 2011]. Last checked: 28. November 2011].

[Wik11b] Wikipedia. Suchmaschine. URL: `http://de.wikipedia.org/wiki/Suchmaschine`, [Stand: 28. November 2011]. [Last checked: 28. November 2011].

[Wik12] Wikipedia. Textkorpus. URL: `http://de.wikipedia.org/wiki/Textkorpus`, [Stand: 27. Januar 2012]. [Last checked: 27. Januar 2012].

[WP10] Martin Wiesner and Daniel Pfeifer. Adapting recommender systems to the requirements of personal health record systems. In *Proceedings of the 1st ACM International Health Informatics Symposium*, IHI '10, pages 410–414, New York, NY, USA, 2010. ACM.

[yah12] yahoo. Boos api. URL: `http://developer.yahoo.com/search/boss/`, [Stand: 24. Januar 2012]. [Last checked: 24. Januar 2012].

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

(Ort, Datum) (Unterschrift)