Ostfalia
Hochschule für angewandte Wissenschaften

Fakultät Informatik

Alexander Reinhard Mengel, 70453431

Untersuchung von Teilbereichen des Internets im Hinblick auf mögliche Gruppierungen basierend auf diskreten Merkmalen

Abschlussarbeit zur Erlangung des Hochschulgrades

Master of Science (M.Sc.)

im Studiengang Medieninformatik Online an der

Ostfalia Hochschule für angewandte Wissenschaften

– Hochschule Braunschweig/Wolfenbüttel

Erste/-r Prüfer/-in: Prof. Dr.-Ing. Nils Jensen

Zweite/-r Prüfer/-in: Prof. Dr. rer. nat. habil Torsten Sander

Eingereicht am 21.05.2022

Zusammenfassung & Abstract

Zusammenfassung

Das Ziel der gegenwärtigen Masterarbeit ist die Untersuchung von Metadaten von Ressourcen (hier zunächst XML- sowie HTML-Dokumente) im Hinblick darauf, mittels dieser Metadaten später sinnvolle Gruppierungen auf ihnen vornehmen zu können um auf diesem Weg Gruppen bzw. Klassen in ihnen zu entdecken.

Um die Untersuchung auf einer möglichst repräsentativen Stichprobe durchzuführen, wird zunächst das Thema der Statistik und Stichprobenerhebung behandelt. Ebenfalls wird im Folgenden zunächst die Theorie und dann die praktische Umsetzung in Form dreier Webcrawler realisiert, welche eine unvoreingenommene Erhebung ermöglichen sollen.

Anschließend werden grundsätzliche Elemente der Möglichkeiten sowie Grenzen von Gruppierung und Clustering behandelt.

Unter Berücksichtigung dieser Erkenntnisse werden die gewonnenen Stichproben begutachtet, um zu Erkenntnissen über ihre Möglichkeiten zur Gruppierung zu gelangen.

Schlüsselbegriffe: Gruppierung, Clustering, Crawling, Metadaten, Web-Metadaten

Abstract

Target of this master thesis ist the inspection of metadata from ressources (in this case XML and HTML documents) with the aim in mind to find soundful groupings among them, to discover and extract soundful groups or clusters among them.

First statistics and sampling is discussed to ensure that the further analysis of the metadata can be generalized. With the aim to generate and unbiased sample, theory of webcrawling is discussed. Three webcrawlers with different strategies are developed.

In the next part, foundational elements of groupings and clusterings are discussed.

With help of this knowledge, the samples generated by the webcrawlers are analyzed to decide whether their metadata can be helpful for he question of grouping.

Keywords: grouping, clustering, crawling, metadata, web metadata

1 Inhaltsverzeichnis

2 Abbildungsverzeichnis

3 Tabellenverzeichnis

1 Einführung

In der Informatik und insbesondere in statistischen Verfahren bzw. Methoden des maschinellen Lernens besteht ein grundsätzliches Interesse daran, innerhalb von Datenbeständen existierende Muster zu erkennen. Diese können dabei helfen, mehr über die eigentlichen Daten bzw. ihre Beziehungen untereinander zu verstehen. Im Rahmen dieser Arbeit wird untersucht, anhand welcher Metadaten sich Webseiten in Form von XML- bzw. HTML-Dokumenten des Internets mit der Top-Level-Domain *.de* gruppieren lassen können.

Das Ziel besteht darin, Informationen über das gesamte Internet bzw. genauer des World Wide Webs mit der Top-Level-Domain *.de* zu gewinnen. Da eine Untersuchung sämtlicher Ressourcen in Form von XML- bzw. HTML-Dokumenten nicht praktikabel ist, soll nur ein kleiner Ausschnitt daraus tatsächlich untersucht werden. Um dennoch eine möglichst hohe Repräsentativität für das deutsche WWW in Form von XML- bzw. HTML-Dokumenten mit der Top-Level-Domain *.de* zu erreichen, wird ausgehend von einigen thematisch verschiedenen Webseiten aus Bereichen wie Sport oder Handel zu anderen Webseiten gefolgt. Dieser Prozess wird wiederholt angewendet. Die initialen Webseiten weisen dabei jeweils hohe Besucherzahlen auf. Dadurch sollen sie eine gewissen Repräsentativität für ihren jeweiligen thematischen Bereich darstellen. Genauere Informationen werden im Kapitel über Realisierung der eigenen Ideen dargelegt. Da im Internet insgesamt ein hoher Vernetzungsgrad herrscht, kann angenommen werden, dass dadurch eine hohe Repräsentativität der so gewonnen Stichprobe bezüglich des gesamten World Wide Webs mit der Top-Level-Domain *.de* erreichbar ist.

Ein Teil der Arbeit besteht darin zu untersuchen, welche Merkmale erhoben werden können und für die Fragestellung geeignet sind. Dabei können sowohl metadatenbasierte Merkmale wie beispielsweise die verwendete Landessprache bzw. verwendete MIME-Types oder die Anzahl der eingebundenen Bilder erhoben werden als auch inhaltsbasierte Merkmale wie etwa die relevantesten Schlagwörter, welche man aber ebenfalls auch als Metadaten auffassen kann. Hierbei soll geprüft werden, welche der Merkmale geeignet sind, um inhaltliche Aussagen über die Webseiten treffen und so auch eine inhaltsbasierte Gruppierung vornehmen zu können.

Für die Erhebung der Merkmale soll ggf. auch ein Parser wie beispielsweise *Apache Tika* genutzt werden. Mit einem solchen Parser können einfach die Metadaten erhoben werden, zudem existieren Erweiterungen für eine inhaltsbasierte Untersuchung.

Für den zu untersuchenden Ausschnitt sollen sämtliche relevante Merkmale sowie der durch die Hyperlinks entstandene Verlinkungsgraph - in einer geeigneten durchsuchbaren Form - abgespeichert werden.

1.1 Zu untersuchende Fragestellungen

Auf dem so gewonnen Datenbestand werden dann die nachfolgenden Fragestellungen untersucht:

- Welche Merkmale eignen sich am besten, um die Gruppierung der Webseiten möglichst *disjunkt* zu gestalten, also so, dass jede Webseite möglichst nicht mehr als einer Gruppe angehört?
- Welche Merkmale eignen sich am besten, um eine möglichst *totale* Aufteilung der Webseiten in Gruppen zu erreichen, also eine Wahl derjenigen Attribute vorzunehmen, sodass jede Webseite in wenigstens einer durch die Merkmalsausprägungen entstandenen Gruppen passiert?
- Welche Merkmale eignen sich am besten, um die Gruppierung der Webseiten möglichst *disjunkt* und *total* zu gestalten?

- Wie müssten die Merkmale gewählt werden, sodass die entstehenden Gruppen möglichst *gleichmächtig* sind? Also eine möglichst identische Anzahl von Webseiten zu enthalten?

Ebenfalls soll noch untersucht werden:

- Welche der Clusteringalgorithmen können auf die gewonnenen Merkmale angewendet werden?

Falls es die Bearbeitungszeit gestattet:

- Besteht zwischen bestimmten Merkmalen und der Verlinkungsstruktur bzw. des Verlinkungsgraphen eine Korrelation oder sogar Kausalität? Beispielsweise kann hier untersucht werden, ob Webseiten mit gleichen Merkmalen auch direkt miteinander verlinkt sind. Geht die direkte Verlinkung lediglich einher mit der Gleichheit der Merkmale (=Korrelation)? Oder ist das jeweils Eine statistisch ursächlich für das Andere (=Kausalität)?

- Ebenfalls könnten später noch Metriken erdacht bzw. untersucht werden, die nicht auf diskreten Merkmalen basieren, sondern auf Wahrscheinlichkeiten.

1.2 Vorgehensweise

Um eine möglichst generelle Aussage über WWW mit der Top-Leve-Domain .de gewinnen zu können, werden zunächst statistische Grundlagen behandelt. Um zudem möglichst repräsentative Stichproben von Ressourcen (hier zunächst XML- sowie HTML-Dokumente) erlangen zu können, sollen diese selbstständig erhoben werden. Dazu wird in die Theorie der Crawler eingeführt und anschließend werden 3 Variationen von Crawlern entwickelt, welche mit verschiedenen Suchstrategien arbeiten. Die Idee ist, durch einen Vergleich der von ihnen erhobenen Ressourcen hinsichtlich der auf diesen Ressourcen relevanten Metadaten generalisierte Aussagen über die Metadaten aller Ressourcen des WWW mit der Top-Leve-Domain .de gewinnen zu können. Anschließend wird die Theorie und grundsätzliche inhärente Grenzen der Gruppierung bzw. des Clusterings diskutiert. Unter Berücksichtigung dieser Erkenntnisse werden schließlich die gewonnen Metadaten der Stichproben begutachtet um zu Aussagen über die obigen Forschungsfragen zu gelangen.

2 Statistische Grundbegriffe sowie Grundlagen

Die folgende Einführung in Grundlagen der Statistik nimmt Bezug auf das Werk *Statistik von Fahrmeier* (vgl. Fahrmeir et al. 2016: 10–17). Die Grundbegriffe sind für die folgende Arbeit wichtig, da die Repräsentativität der zu gewinnenden Stichprobe relevant für die darauf aufbauende Analyse ist.

Allgemein kann man die Statistik als Werkzeug zur Datenanalyse beschreiben. Ihre Methoden sind überall dort notwendig, wo durch empirische Untersuchungen Daten erhoben und analysiert werden sollen. So liefert sie Methoden darüber, wie wissenschaftliche Untersuchungen zu planen sind, ihre Ergebnisse zu analysieren und interpretieren sind.

Man kann die Statistik unterteilen gemäß ihrer hauptsächlichen Aufgabengebiete in die *deskriptive Statistik*, *explorative Statistik* sowie *induktive Statistik*.

2.1 Deskriptive Statistik

Dieser Teil behandelt die Beschreibung und Darstellung bzw. Aufbereitung von Daten, ebenfalls geht es um die Kompression von Daten bzw. den darin enthaltenen Informationen. Daten äußern sich in Form von Ausprägungen bzw. Werten von Merkmalen bzw. Variablen.

Merkmale können nach verschiedenen Kriterien charakterisiert werden. So kann die Ausprägung des Merkmals *Alter in Jahren* bei einer bestimmten, eindeutig definierten Person *23* lauten. Je nach der zu untersuchenden Fragestellung existieren Variablen, die durch andere Variablen beeinflusst werden, diese werden auch *Zielgrößen* genannt. Zielgrößen werden beeinflusst durch beobachtbare d.h. messbare Variablen, welche *Faktoren* bzw. *Einflussgrößen* genannt werden sowie durch nicht beobachtbare und damit nicht messbare Variablen. Innerhalb der Menge der nicht beobachtbaren Variablen kann weiter unterschieden werden zwischen für eine Analyse wichtige Faktoren, die auch *latente Größen* bezeichnet werden sowie den *Störgrößen*. Ermittelt werden soll hier, welche der Variablen Zielgrößen wie beeinflussen. Beispielsweise kann bei einer Untersuchung das Gehalt eine Zielgröße sein und Einflussgrößen etwa die fachspezifische und allgemeine Berufserfahrung, die Art des Bildungsabschlusses und Ähnliches. Relevante Störgrößen könnten hier die nicht öffentlich einsehbaren Präferenzen des potenziellen Arbeitgebers oder auch des Arbeitnehmers sein. Da diese Informationen fehlen bzw. Störgrößen oder auch latente Größen existieren, kann mittels der erhobenen Einflussgrößen die Zielgröße *Gehalt* nur bestenfalls approximativ bestimmt werden.

Ein weiteres Aufgabenfeld innerhalb der deskriptiven Statistik ist die *Datenvalidierung*. Durch die Datenaufbereitung können beispielsweise Fehler, die etwa bei der manuellen Übertragung von Daten von einem Fragebogen in ein Computerprogramm aufgetreten sind, leicht identifiziert und damit korrigiert werden.

In der deskriptiven Statistik werden noch keine auf der Wahrscheinlichkeitstheorie basierende Methoden eingesetzt. Damit ist kein formaler Rückschluss auf Daten und Objekte möglich, die außerhalb des Teils der Erhebung liegen.

2.2 Explorative Statistik

Hier geht es allgemein um das Auffinden von Strukturen bzw. Besonderheiten in Daten, auch *Exploration* genannt, um anschließend daraus neue Fragestellungen bzw. Hypothesen zu gewinnen.

Ebenfalls finden hier Methoden basierend auf der Wahrscheinlichkeitstheorie noch keinen Eingang, insgesamt sind aber die Methoden inhaltlich stärker von denen der induktiven Statistik inspiriert.

Die explorative Statistik findet häufig dann Anwendung, falls Fragestellungen nicht exakt vorab gefasst sind. Methoden auf dieser Ebene sind oft rechenaufwendig, daher sind leistungsstarke Rechner eine wichtige Voraussetzung. Auch wenn keine Wahrscheinlichkeitsaussagen über Objekte außerhalb der Erhebung hinaus getroffen werden können, so ergeben sich häufig ausgeprägte Hinweise bzw. empirische Evidenz für bestimmte Muster in den Daten.

2.3 Induktive Statistik

In diesem Teil geht es um das Ziehen von statistischen Schlüssen mittels stochastischer Modelle.

In der induktiven Statistik werden Methoden der Wahrscheinlichkeitstheorie eingesetzt, um auch zu Aussagen über die Objekte zu gelangen, die kein Teil der Erhebung waren. Es geht darum, die anhand der Erhebung gewonnen Erkenntnisse induktiv verallgemeinern zu können. Bezugnehmend auf das angeführte Beispiel mit dem Gehalt könnte man, wenn man mehrere Daten zu Gehältern gesammelt hat, eine Aussage über das durchschnittliche Gehalt in einem räumlich und zeitlich abgegrenzten Bereich treffen.

2.4 Grundbegriffe der Statistik

Als Fachwissenschaft verfügt die Statistik über eine eigene Fachterminologie, die nachfolgend kurz umrissen werden soll.

Statistische Einheiten sind die Objekte, an denen man Daten beobachten möchte. Dabei kann es sich um Personen oder auch abstrakte Konstrukte handeln. Die Zusammenfassung aller statistischer Einheiten, genau definiert bezüglich der räumlichen, zeitlichen sowie inhaltlichen Aspekte, bildet die *Grundgesamtheit* bzw. *Population*. Die Grundgesamtheit ist damit die Menge aller statistischer Einheiten, über die man zu Erkenntnissen gelangen möchte. Die Anzahl der Elemente der Grundgesamtheit kann endlich groß oder auch (potenziell) unendlich groß sein. Handelt es sich bei den statistischen Einheiten beispielsweise um die Ausfallzeiten einer vollautomatischen Fertigungsstraße innerhalb der Betriebszeiten, so existieren potenziell unendlich viele Objekte dieser Art. Möchte man nur eine Teilmenge der Grundgesamtheit untersuchen, so nennt man diese *Teilgesamtheit* bzw. *Teilpopulation*. Die später tatsächlich untersuchte Teilpopulation nennt man *Stichprobe*. Im Idealfall stellt sie ein angemessenes Abbild der Grundgesamtheit dar. Dies kann beispielsweise dann gegeben sein, falls jede statistische Einheit eine identische Chance besitzt, Teil der Stichprobe zu werden. Die statistischen Einheiten der Stichprobe besitzen i.d.R. mehrere *Merkmale*. Die *Werte*, die diese Merkmale bei ihnen annehmen, werden untersucht, man bezeichnet sie auch als *Ausprägungen* bzw. *Merkmalsausprägungen*.

2.5 Arten von Merkmalen

Man kann Merkmale aufgrund verschiedener Eigenschaften unterteilen. Diese Unterteilungen sind relevant, weil sie festlegen, welche späteren Operationen im Rahmen statistischer Methoden mit ihnen durchgeführt werden dürfen. Darauf basiert auch die Wahl der möglichen Clusteringverfahren.

2.5.1 Unterteilung von Merkmalen anhand der Anzahl von Ausprägungen

Falls ein Merkmal endlich viele Ausprägungen oder abzählbar unendlich viele Ausprägungen umfasst, so heißt es *diskret*. Falls ein Merkmal alle Ausprägungen auf kontinuierlichem Intervall annehmen kann, heißt es *stetig*.

Zusätzlich existieren als eine Zwischenform *quasi-stetige* Merkmale. Sie entstehen bei diskreter Messung mit jedoch zwischen den Merkmalsausprägungen sehr feinen Abstufungen. Damit werden sie wie stetige Merkmale behandelbar. Außerdem entstehen sie durch Messungenauigkeiten.

Ausprägungen eines stetigen Merkmals können durch Klassen so zusammengefasst werden, dass es als diskret behandelbar ist. Dieses Vorgehen heißt auch *Gruppierung*, *Klassierung* bzw. *Kategorisierung*. In diesem Kontext ist der Begriff Gruppierung aber nicht deckungsgleich mit der in dem Titel dieser Arbeit angegeben Begriff, da die später zu untersuchenden Merkmale bereits diskreter Natur sind.

2.5.2 Unterteilung von Merkmalen anhand der Skalenniveaus

Merkmale werden auf Skalenniveaus gemessen. Die verschiedenen Skalenniveaus stellen eine Ordnung dar und werden aufsteigend sortiert vorgestellt. Prinzipiell kann man Merkmale, die auf einem höheren Skalenniveau gewonnen wurden, so transformieren, dass ihre Ausprägungen niedriger skaliert sind. Dabei gehen jedoch Information verloren. Ein umgekehrtes Vorgehen ist im Normalfall nicht möglich.

Wenn Ausprägungen von Merkmalen Namen bzw. Kategorien sind, handelt es sich um *nominalskalierte* Daten. Hier kann nur die Häufigkeit einer Ausprägung sinnvoll erhoben werden.

Wenn Ausprägungen von Merkmalen geordnet werden können, ohne dass ihre Abstände sinnvoll interpretierbar sind, handelt es sich um *ordinalskalierte* Daten.

Wenn die Abstände von Ausprägungen von Merkmalen sinnvoll interpretiert werden können, handelt es sich um *intervallskalierte* Daten.

Wenn die Abstände sowie Quotienten von Merkmalsausprägungen sinnvoll interpretiert werden können, handelt es sich um *verhältnisskalierte* Daten.

Ausprägungen, die intervallskaliert oder verhältnisskaliert sind, werden auch zusammenfassend *kardinalskaliert* bezeichnet. Sie werden auch als *metrisch* bezeichnet.

In besonderen Fällen können auch Merkmale mit niedrigeren Skalenniveaus einer sinnvollen Berechnung zugeführt werden, die gewöhnlicherweise Merkmalen höheren Skalenniveaus vorbehalten bleibt. So bildet das arithmetische Mittel von Schulnoten, obwohl diese ordinalskaliert sind, eine interpretierbare Maßzahl, die insbesondere sinnvoll vergleichend genutzt werden kann.

2.5.3 Unterteilung von Merkmalen basierend auf ihrer inhaltlichen Art

Merkmale, deren Ausprägungen nur eine Qualität beschreiben, ohne ein Ausmaß von dieser, nennt man *qualitative* bzw. *kategoriale Merkmale*. Sie können dabei maximal ordinalskaliert sein. Ordinalskalierte Merkmale besitzen zwar durch die Anordnung einen größenmäßigen Aspekt, da aber meist der qualitative Aspekt bei Ihnen überwiegt, werden sie den qualitativen Merkmalen zugeordnet.

Merkmale, deren Ausprägungen ein Ausmaß bzw. eine Intensität darstellen, nennt man *quantitative Merkmale*. Kardinalskalierte Merkmale sind damit auch immer quantitative Merkmale.

3 Grundsätzliche Aspekte der Datengewinnung

Im Folgenden geht es um relevante Aspekte der Datengewinnung, erneut bezugnehmend auf das Werk *Statistik von Fahrmeier* (vgl. Fahrmeir et al. 2016: 18–25). Die Informationen sind relevant, um die Entscheidungen bzgl. des Versuchsaufbaus zur Gewinnung von den Stichproben nachvollziehen zu können.

3.1 Empirie

Allgemein werden statistische Verfahren dann eingesetzt, wenn Fragen aus der Forschung unter Berücksichtigung der *Empirie* getestet werden sollen. Eine empirische Fragestellung ist dabei eine zunächst theoretisch erdachte Annahme, die anschließend anhand der Wirklichkeit getestet wird. Um diese Testbarkeit zu ermöglichen, müssen sowohl die verursachenden Einflussgrößen als auch die Zielgrößen messbar sein bzw. dahingehend umgestaltet werden, dass sie final quantitativ erfassbar sind. Dieser Vorgang wird auch *Operationalisierung* genannt.

3.2 Versuchsplan

Bevor mit den eigentlichen Versuchen begonnen wird, muss zunächst ein *Versuchsplan* erstellt werden. Darin wird geklärt, welches Ziel das Experiment verfolgt, wie dieses Ziel erreicht werden kann sowie welche statistischen Methoden geeignet erscheinen, um die dann gewonnen Ergebnisse statistisch zu bekräftigen. Die Festlegung des Ziels erfordert zunächst, die relevante Grundgesamtheit zu definieren und damit abzugrenzen. Wichtig ist ebenfalls, den Umfang der zu gewinnenden Stichprobe festzulegen. Um die Genauigkeit der Stichprobe bestimmen zu können, ist der Einsatz statistischer Verfahren relevant.

3.3 Arten der Datenerhebung

Es existieren verschiedene Arten der Erhebung von Daten. So können Daten bereits vorhanden sein und müssen nur noch erhoben oder aber mittels des Experiments generiert werden. Sie können im Rahmen einer *primärstatistischen* Erhebung gewonnen worden sein, oder aus bereits vorhandenen Daten als *Sekundärstatistik* zusammengefügt worden sein. Unabhängig von der Art der Erhebung stellen fehlende Daten ein Problem dar. Eine geläufige Strategie besteht darin, die gesamte Untersuchungseinheit nicht zu berücksichtigen, dadurch entsteht jedoch ein Verlust an Information.

3.4 Stichproben

Bei der Gewinnung einer Stichprobe soll eine hohe Effektivität sowie Repräsentativität für die Grundgesamtheit in Bezug auf die für eine Untersuchung relevanten Merkmale erzielt werden. Repräsentativität meint hier, dass die Stichprobe ein möglichst getreues Abbild der Grundgesamtheit darstellt. Prinzipiell kann Repräsentativität erreicht werden, indem die verschiedenen Elemente aus der Grundgesamtheit durch einen Zufallsmechanismus Teil der Stichprobe werden können, so dass jede Untersuchungseinheit mit einer Wahrscheinlichkeit ausgewählt werden kann.

Es existieren verschiedene Arten von Stichproben, welche im Folgenden kurz beschrieben werden.

3.4.1 Einfache Zufallsstichprobe

In der *einfachen Zufallsstichprobe* besitzen Teilmengen mit identischer Mächtigkeit der Grundgesamtheit dieselbe Wahrscheinlichkeit gezogen zu werden. Daraus folgt, dass auch jede einzelne Untersuchungseinheit mit identischer Wahrscheinlichkeit gezogen werden kann. Dazu müssen theoretisch sämtliche Elemente der Grundgesamt nummerierbar sein und als Liste vorliegen. Man kann dabei auf einmal alle Elemente der Stichprobe ziehen oder jeweils ein Element und danach neu mischen. Ebenfalls kann man eine systematische Ziehung vornehmen, bei der aus einer geordneten Grundgesamtheit jedes x.-te Element gezogen wird.

3.4.2 Geschichtete Zufallsstichprobe

In der *geschichtete Zufallsstichprobe* zerlegt man die Grundgesamtheit in sich nicht überlappende Schichten, aus diesen werden jeweils zufällig Untersuchungseinheiten gezogen. Es kann hier durch den Schichtungseffekt zu einer insgesamt informativeren Stichprobe kommen, falls die Variable, nach der sich die Schichtung richtet, mit dem eigentlich zu untersuchendem Merkmal korreliert.

3.4.3 Klumpenstichprobe

In der Klumpenstichprobe zerfällt die Grundgesamtheit auf natürlich Weise in Schichten, die dann *Klumpen* genannt werden. Aus diesen Klumpen wählt man einige aus und führt in den ausgewählten Klumpen eine Vollerhebung durch. Diese Stichprobenart ist dann sinnvoll, wenn die Klumpen bzgl. der oder den später zu untersuchenden Variablen heterogen sind, die Verhältnisse dieser Variablen bezüglich der Grundgesamtheit gut repräsentieren und innerhalb der Klumpen eine hohe Homogenität herrscht.

3.5 Weitere Arten von Auswahlverfahren für Untersuchungseinheiten

Neben den obigen Verfahren zur Auswahl von Untersuchungseinheiten kann man auch ein *bewusstes Auswahlverfahren* anwenden. Dieses ist dann sinnvoll, falls man im Vorhinein zu einem oder mehreren Merkmalen der Grundgesamtheit die verhältnismäßige Aufteilung der Ausprägungen kennt. Dieses kann man dann versuchen auch in der Stichprobe beizubehalten, um die Qualität der Stichprobe zu steigern.

Alternativ kann man subjektiv *typische Fälle* als Vertreter für die Grundgesamtheit untersuchen. Da bei der Wahl dieser Fälle relevante Kriterien übersehen werden können, kann dieses Vorgehen problematisch sein.

3.6 Studiendesigns

Während man bei *Zeitreihen bzw. Längsschnittstudien* ein bzw. mehrere Untersuchungseinheiten über einen Zeitraum beobachtet, werden bei einer *Querschnittstudie* zu einem Zeitpunkt bzw. eng gefassten Zeitraum ein bzw. mehrere Untersuchungseinheiten bzgl. ihrer Merkmale beobachtet.

4 Zur Theorie und Praxis des Crawlerdesigns anhand des Mercator Crawlers

In diesem Abschnitt sollen wesentliche theoretische sowie praktische Aspekte des Designs sowie der Implementierung von Webcrawlern erläutert werden.

4.1 Allgemeiner Hintergrund

Die Inhalte sind für die Arbeit relevant, da die Stichprobe mittels eines selbst entwickelten Prototyps eines Webcrawlers erhoben werden, um die Parameter und Strategie der Stichprobenerhebung selbst steuern zu können. Als Beispiel dient hier primär das Design des *Mercator* Webcrawlers in seiner ursprünglichen Form (vgl. Heydon/Najork 1999).

Die im Folgenden angeführten grundlegenden Designs sowohl von Mercator, der *Google Search Engine* sowie des *Internet Archive Carwlers* sind jeweils bereits über 20 Jahre alt und in der Zwischenzeit weiter ausdifferenziert wurden. Da sie aber die immer noch gültigen Kernkomponenten und grundlegenden Designentscheidungen widerspiegeln, welche bei dem Design eines Webcrawlers zu beachten sind, werden sie hier dargestellt. Später werden zudem einige der wichtigsten Herausforderungen beschriebenen, die dem Webmining inhärent sind.

Die Hauptziele bei der Entwicklung von Mercator bestanden in einer guten *Skalierbarkeit* sowie *Erweiterbarkeit:*

Skalierbarkeit meint hier, dass man potenziell das ganze Web bzw. große Teile davon crawlen können sollte. Dies soll erreicht werden, indem Datenstrukturen so implementiert werden, dass nur ein fest begrenzter Teil des Arbeitsspeichers genutzt wird, unabhängig von der Größe des gesamten Crawls. Daraus folgt, dass der größte Teil der Datenstrukturen sich auf der Festplatte bzw. dem persistenten Speicher befindet und nur kleine Teile aus Gründen der Effizient im Arbeitsspeicher verbleiben.

Erweiterbarkeit meint, dass das Crawlerdesign modularisiert ist und durch 3. Parteien einfach um eigene Funktionalitäten erweitert werden kann, beispielsweise um den Crawler um eine Random-Walk Strategie zu erweitern.

4.2 Hauptkomponenten eines skalierbaren Webcrawlers

Die folgende Abbildung verdeutlicht die wesentlichen Module eines skalierbaren Webcrawlers. Dabei beginnt der Vorgang mit einer initialen Menge von URLs als Ausgangsbasis für den Crawl.

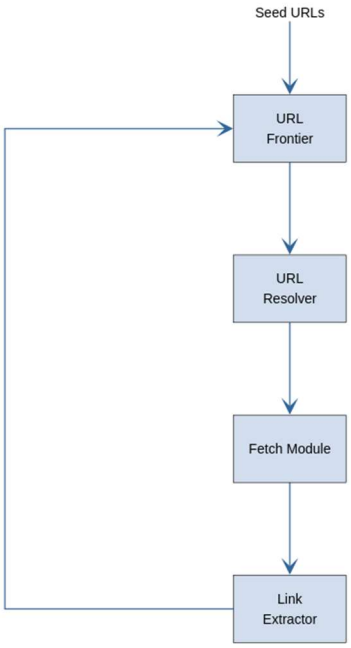

Abbildung 1: Hauptkomponenten eines Webcrawlers

Diese URLs werden in der *URL Frontier* gespeichert, einer Datenstruktur, in der die jeweils zukünftig zu crawlenden URLs vorgehalten werden. Es existieren verschiedene Implementierungen, etwa als einfache Queue, als Queue mit Prioritäten oder auch als Queue, die sich ihrerseits speist aus Website spezifischen Queues. Anschließend werden die URLs nach und nach aus der URL Frontier entfernt und dem *URL Resolver* zugeführt, welcher die Hostnamen in IP-Adressen übersetzt, der dafür notwendige Dienst wird *Domain Name System* genannt. Daraus ergibt sich, von welchem Webserver das Dokument herunterladen werden kann. Das *Fetch Modul* lädt das Dokument schließlich herunter, typischerweise über *HTTP*. Anschließend werden sämtliche Links durch den *Link Extractor* herausgefiltert, dabei werden ggf. vorkommende relative Links in absolute Links umgewandelt. Absolute Links beginnen immer zunächst mit einem Schema, welches das Protokoll bestimmt, beispielswiese HTTP oder FTP. Falls die so gewonnenen Links neu sind bzw. nicht bereits besucht wurden, werden sie der URL Frontier hinzugefügt.

4.3 Design des Mercator Webcrawlers

Im Folgenden wird das Design des Webcrawlers Mercator genauer dargestellt aufgrund seines Einflusses für skalierbare sowie erweiterbare Webcrawler sowie seiner transparenten Dokumentation.

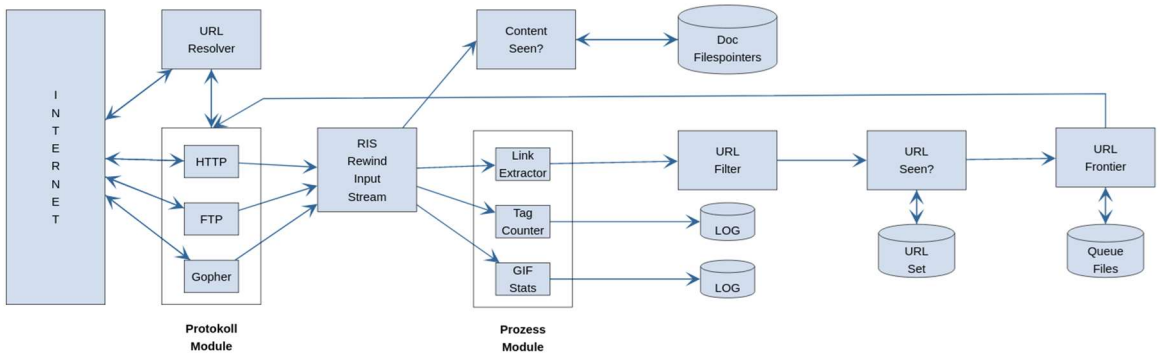

Abbildung 2: Mercator Design

Es findet ein paralleles Crawling mit mehreren hundert Worker Threads statt. Eine der wesentlichsten Herausforderungen besteht in der Gestaltung der Datenstrukturen, so dass ein wohl balancierter *Time-Memory Tradeoff* existiert. Der Tradeoff besteht hier zwischen den entgegengesetzten Zielen eines möglichst geringen (Arbeits)speichergebrauchs bzw. allgemeiner dem möglichst minimalen Konsum von Raum einerseits sowie einer möglichst hohen Performanz bzw. einer möglichst geringen Zeit, die für das Absolvieren einer Rechenaufgabe benötigt wird.

4.3.1 URL Frontier

Mercator nutzt als Suchstrategie standardmäßig die Breitensuche, implementiert mittels einer *FIFO Queue*. Ein mögliches Problem kann darin bestehen, dass mehrere parallele HTTP Requests zum selben Webserver aufgebaut werden könnten, was zur Folge haben kann, dass der Webserver die Antwortzeit erhöht oder die Requests gar nicht mehr beantwortet. Es muss also bei der Übermittlung von URLs aus der URL Frontier hin zu den Protokoll Modulen eine angepasste Strategie verwendet werden. Die Lösung besteht darin, dass nicht immer der Kopf der Queue zurückgegeben wird, sondern ggf. eine URL nahe beim Kopf der Queue, von der sichergestellt ist, dass ihr Host momentan keine ausstehenden Requests im Rahmen der Crawlvorgangs besitzt. Jeder Worker Thread besitzt für die von ihm zu crawlenden URLs je *kanonischem Hostnamen* eine eigene FIFO Subqueue. Falls nun aus der URL Frontier eine neue URL übermittelt wird, wird zunächst ihr kanonischer Hostname ermittelt und dann geprüft, ob bereits in einem der Worker Threads eine Subqueue für diesen Hostnamen existiert und in diesem Fall dort angefügt, ansonsten kann für sie eine neue Subqueue angelegt werden, beispielsweise bei dem Worker Thread mit den gegenwärtig wenigsten noch zum Crawlen ausstehenden URLs.

4.3.2 Protokoll Modul

Basierend auf dem Schema der von der URL Frontier übermittelten URL wird das entsprechende Protokoll Modul ausgewählt und das darin enthaltene Fetch Modul aufgerufen, um die Daten nach Übersetzung in eine IP-Adresse aus dem Internet abzurufen. Das http Protokoll Modul berücksichtigt zudem das *Robots Exclusion Protocol*, in dem mittels eines Dokuments angegeben werden kann, welche Teile einer Website gecrawlt werden dürfen. Daher wird zunächst diese Information für eine Webseite abgerufen und in Abhängigkeit davon die erlaubten Teile gecrawlt. Um unnötigen Traffic zu vermeiden, werden die Informationen aus dem Robots Exclusion Protocols für die zuletzt besuchten Webseiten in einem Cache vorgehalten und vor jedem neuen Informationsabruf zunächst in dem Cache geprüft, ob einschränkende Informationen darüber vorhanden sind. Außerdem verfügt das HTTP Protokoll Modul über eine Time-Out Funktionalität, um zu verhindern, dass ein bösartiger Webserver den Crawler auf unbegrenzte Zeit gefangen halten kann.

4.3.3 RIS – Rewind Input Stream

In dem Rewind Input Stream werden abgerufenen Ressourcen teilweise oder auch vollständig in einem Cache vorgehalten, um sie daraufhin, falls es sich um neue Ressourcen handelt, einem oder auch mehreren Prozess Modulen zur Verfügung zu stellen. Für diesen Stream kann auch eine mögliche maximale Ressourcengröße definiert werden als Schutz gegen bösartige Webserver, die ggf. Ressourcen von praktisch unbegrenzter Größe ausliefern könnten. Ebenfalls umfasst der Stream Methoden für das *Lexing* bzw. *Tokenizer*, um MIME-Typen spezifische Parser bauen zu können. Je Worker Thread existiert ein eigener Rewind Input Stream.

4.3.4 Content Seen Test

Ein typisches Merkmal des Webs besteht darin, dass ein großer Teil der Informationen identisch bzw. beinahe identisch ist. Beispielsweise können verschiedene URLs auf eine identische Ressource auf demselben Server verweisen. Außerdem werden häufig Ressourcen gespiegelt und existieren damit auf jeweils eigenen Servern, die auch über jeweils eigene Hostnamen verfügen, dennoch zueinander identische Ressourcen besitzen.

Würde man die vollständigen Inhalte einer jeden bereits gecrawlten Ressource speichern um anschließend bei einer Ressource aus dem RIS zu prüfen, ob sie bereits bekannt ist, so wäre ein solcher Test sehr raum- sowie zeitkostenintensiv und praktisch nicht durchführbar. Stattdessen wird für jede gecrawlte Ressource eine 64-bit lange Prüfsumme gebildet und in einem *Ressource Fingerprint Set* abgespeichert. Diese Prüfsumme wird durch den Fingerprinting Algorithm von Rabin gebildet. Dieser besitzt gegenüber den Verfahren MD5 sowie SHA-1 die Eigenschaft, dass mit einer sehr hohen Wahrscheinlichkeit inhaltlich verschiedene Ressourcen auch verschiedene Fingerprints erhalten bzw. umgedreht gilt, dass mit hoher Wahrscheinlichkeit zu einem Fingerprint auch nur genau eine Ressource existiert, die Eigenschaft von Funktionen wird auch Injektivität genannt. Da auch jedes Element aus der Menge der Fingerprints ein Bild ist, folgt, dass die durch den Algorithmus von Rabin gebildete Funktion mit hoher Wahrscheinlichkeit auch bijektiv bzw. eineindeutig ist.

Falls das gebildete Fingerprint Set insgesamt zu groß für den Arbeitsspeicher wird, verbleibt nur eine kleine Hash-Tabelle darin, der größere restliche Teil wird dann in Form einer sortierten Liste auf der Festplatte bzw. dem persistenten Speicher abgelegt. Ebenfalls ist es nicht sinnvoll, einen größeren Cache für die zuletzt an das Fingerprint Set gestellten Requests vorzuhalten, da zwischen je zwei Requests, die an das Fingerprint Set gestellt werden keine hohe Lokalitätseigenschaft besteht, die Wahrscheinlichkeit also nicht hoch ist, dass ein gestellter Request sich zeitnah wiederholen wird.

4.3.5 Prozess Module

Falls der Content Seen Test negativ ausgefallen ist, werden die Ressourcen durch eines bzw. mehrere der Prozess Module verarbeitet. Die Entscheidung darüber basiert auf den ermittelten MIME-Typen. So können beispielsweise Statistiken für die Anzahl von GIFs oder auch Tags generiert werden bzw. eingebettete Links extrahiert werden.

4.3.6 URL Filter

Bevor eine URL von dem Worker Thread zur nächsten Prüfung weitergeleitet wird, werden hier die URLs gefiltert. Die Filterung kann basieren auf der Domain, dem URL-Präfix und der Protokollart. Die verschiedenen

Filter können mit logischen Operatoren verbunden werden. Beispielsweise können hier als bösartig bekannte URLs entfernt werden.

4.3.7 URL Resolver

Der URL Resolver sitzt zwischen den Protokoll Modulen und dem Internet. Sollen Ressourcen aus dem Internet angefordert werden, so werden ihre URLs mittels des Domain Namen Service zunächst in IP-Adressen übersetzt. Dieser Vorgang kann sich bei einem größeren Crawl zu einem Flaschenhals entwickeln, insbesondere, falls DNS genutzt wird, um die Hostnamen von neu entdeckten URLs vor dem URL Seen Test zu kanonisieren. In vielen Implementierungen ist dieser Vorgang synchronisiert und findet damit sequentiell statt. Da man auch von einer hohe Cache Miss Rate ausgehen kann, die neuen URLs also relativ häufig nicht in dem Cache über die letzten Anfragen gefunden werden, entwickelt sich hieraus der genannte Flaschenhals. Eine Lösungsmöglichkeit, die hier genutzt wird, besteht in einem Multi-Threaded URL Resolver, der asynchron arbeitet.

4.3.8 URL Seen Test

Der überwiegende Teil der kanonischen URLs wird aufgrund der großen Anzahl auf Festplatte gespeichert. Um weiterhin Platz zu sparen, werden die URLs dort nicht in textueller Form abgespeichert, sondern als Checksummen fester Länge. Um die Anzahl der Leseoperationen auf der Festplatte zu begrenzen, wird die Lokalitätseigenschaft genutzt, nach der besonders populäre URLs in dem Arbeitsspeicher gehalten werden. Da tendenziell auf eine relativ kleine Menge von populären URLs häufig verwiesen wird, existiert hier eine hohe *in-memory hit rate*. Die Erfahrung mit Mercator hat gezeigt, dass hier im Rahmen eines Crawls aufgrund der ausgeprägten Lokalitätseigenschaft wesentlich weniger Festplattenaufrufe als bei dem Content Seen Test benötigt werden.

Eine weitere relevante Lokalitätseigenschaft, die ausgenutzt werden kann, ist die *Host Name Locality*. Nach ihr kommt es häufig vor, dass URLs auf Ressourcen auf demselben Server verweisen. Diese Lokalitätseigenschaft wird ausgenutzt, indem sie in eine Zugriffslokalitätseigenschaft der URLs auf der Festplatte übersetzt wird. Dies wird erreicht, indem die Checksumme des Hostnames der URL mit der Checksumme der vollständigen URL zusammengeführt wird. Daraus folgt, dass Prüfsummen von URLs auf demselben Server numerisch näher beieinander sind und damit schneller abgerufen werden können.

Als alternative Implementierung verwendet der Internet Archive Crawler einen *Bloomfilter*, eine probabilistische Datenstruktur, die für die Prüfung auf die Mengenzugehörigkeit eines Elements. Aufgrund seiner Eigenschaften kann es hier auch zu einem falsch positiven Ergebnis kommen, falsch negative Antworten sind dagegen ausgeschlossen. Tendenziell skaliert diese Implementierung jedoch schlechter.

4.3.9 Synchrones und asynchrones Lesen und Schreiben

Um parallele Downloads durchführen zu können existieren zwei grundsätzlich verschiedene Strategien. Praktisch können auch Mischformen daraus existieren. Nach dieser Aufteilung besteht eine Strategie darin, den Crawlprozess *single-threaded mit asynchronem I/O* zu implementieren. Folgt man diesem Ansatz, so gestaltet man das Programm von Grund auf so, dass eine Skalierung auf mehrere Maschinen leicht realisierbar ist. Dieser Ansatz wird von den Crawlern von Google sowie dem Internet Archive genutzt. Alternativ zu dieser Strategie kann man den Crawlprozess auch *multi-threaded mit synchronem I/O* implementieren. Dies hat eine tendenziell einfachere Programmstruktur zur Folge, da der Wechsel zwischen den Threads durch den *OS Thread Scheduler* vorgenommen wird. Diese Strategie verwendet Mercator. Eine

solche Strategie ist ebenfalls relativ leicht adaptierbar, so dass Mercator auf mehreren Maschinen laufen kann, wobei weiterhin ein synchrones I/O und mehrere Threads in jedem Prozess laufen.

4.3.10 Sichern des Zustands – Checkpoints

Mercator schreibt regelmäßig Snapshots seines Zustands auf die Festplatte, diese Snapshots sind unabhängig von den anderen im persistenten Speicher befindlichen Datenstrukturen. Ein unterbrochener oder etwa aufgrund eines Fehlers abgebrochener Crawlvorgang kann dann ausgehend von dem letzten Snapshot wieder aufgenommen werden. In regelmäßigen Abständen, beispielsweise alle 24h, bekommt der Main Thread einen Write Lock, damit läuft er isoliert und kann seinerseits die Checkpointmethoden in den einzelnen Modulen aufrufen.

4.3.11 Erweiterbarkeit von Mercator

Ein wesentliches Designmerkmal von Mercator besteht in der Erweiterbarkeit. Darunter verstehen die Entwickler zum einen, dass leicht neue Protokoll Module hinzugefügt werden können, um Ressourcen gemäß zukünftiger neuartiger Netzwerkprotokolle herunterladen zu können, als auch, dass leicht neue Prozess Module hinzugefügt werden können, um die heruntergeladenen Ressourcen individuell angepasst weiterzuverarbeiten. Zum anderen bedeutet dies, dass Mercator leicht umkonfiguriert werden kann, so dass andere Versionen seiner wesentlichen Komponenten zum Einsatz kommen.

Technisch wird die leichte Erweiterbarkeit von Mercator primär dadurch realisiert, indem die Schnittstellen der Komponenten des gesamten Crawlers wohldefiniert sind. In Mercator ist dazu die Schnittstelle jeder Komponente durch eine abstrakte Klasse spezifiziert, von denen auch zumindest einige der Methoden abstrakt sind. Von jeder neuen Komponente wird gefordert, dass sie abgeleitete Klasse dieser jeweils abstrakter Klasse darstellt und die abstrakten Methoden implementiert. Ebenfalls ist eine bereits bestehende Infrastruktur in Form von Bibliotheken mit nützlichen Funktionen für crawltypische Problemstellungen vorhanden. Das konkrete Verhalten des Crawlers wird zudem durch eine Konfigurationsdatei definiert, welche zum Programmstart eingelesen wird. In ihr wird spezifiziert, welche jeweiligen Protokoll und Prozess Module zu nutzen sind, sowie welche der konkret implementierten anderen Komponenten. Durch einen dynamischen Mechanismus werden die entsprechenden Klassen geladen und ihre Instanzen an den entsprechenden Stellen innerhalb der Datenstrukturen eingefügt.

Beispielsweise umfasst die abstrakte *Protocol* Klasse die Methoden *fetch* und *newURL*. Während die fetch Methode für das Herunterladen der Ressource sorgt, leistet die newURL Methode das Parsen des Strings, um darauf ein strukturiertes URL Objekt zu gewinnen. Dies ist relevant, da die URLs in Abhängigkeit des Netzwerkprotokolls unterschiedlich aufgebaut sind. Mittels der abstrakten *Analyzer* Klasse können konkrete Prozess Module implementiert werden, mit denen sich beispielsweise verschiedene Statistiken gewinnen lassen können.

Ebenfalls kann die URL Frontier alternativ implementiert werden. Falls beispielsweise die Anzahl der Host begrenzt ist, etwa bei dem Crawlvorgang eines kleineren Intranets, kann ein Verhalten implementiert werden, nachdem garantiert jeder Worker Thread auch mindestens einen Host bearbeitet, solange die Anzahl der Host die der Worker Threads übersteigt. Dabei bleibt die Einschränkung bestehen, dass ein Host zu einem Zeitpunkt nur durch maximal einen Worker Thread erforscht werden darf.

Durch eine alternative Implementierung der URL Frontier sowie einiger weiterer Komponenten kann Mercator zudem als *Random Walker* fungieren. In einem Random Walk wird aus einer initialen Menge von URLs zufällig eine URL gewählt, die auf dieser Ressource befindlichen Links extrahiert, unter diesen zufällig eine neue URL

als Ziel gewählt und dieser Vorgang rekursiv so lange wiederholt, bis man auf eine Ressource stößt, die selbst keine URLs enthält. In diesem Fall wird der Vorgang wiederholt, wobei wieder eine zufällige URL aus der Menge von URLs gewählt wird. Die initiale Menge von URLs wird dabei um die neu entdeckten URLs aus dem ersten Crawlvorgang erweitert. Je nach konkreter Implementierung des Random Walks kann eine Ressource nur einmal oder auch mehrmals besucht werden.

4.4 Herausforderungen und Hindernisse des Crawlens

Da es sich bei dem World Wide Web um ein dynamisches und hochgradig vernetztes System handelt, welches zudem auch aus Akteuren mit teilweise unethischen Interessen besteht, welches technisch ursprünglich aber auf dem Paradigma eines offenen, wissenschaftlichen Diskurses ohne negative Absichten beruht, erwachsen verschiedene Herausforderungen an einen Crawler.

Während die Anzahl der statischen Ressourcen des Webs, die also nicht dynamisch generiert werden, endlich ist, existieren potentiell unendlich viele abrufbare URLs. Einige der Ursachen dafür liegen in teilweise bereits erwähnten URL Aliasen, Session IDs und Crawlerfallen.

4.4.1 URL Aliase

Je zwei URLs sind Aliase füreinander, falls sie sich auf dieselben Inhalte beziehen. Es existieren verschiedene Ursachen für das Existieren dieser Aliase. So gibt es Host Name Aliase, welche auftreten, wenn mehrere verschiedene Hostnamen mit derselben IP-Adresse korrespondieren. In diesem Fall kann das identische Dokument wenigstens unter jeder der URLs für diesen Hostnamen erreicht werden sowie unter ihrer IP-Adresse. In Mercator findet daher eine Kanonisierung der URL statt für eine entsprechende Anfrage an den DNS, es wird der ggf. vorhandene kanonische Hostname zurückgeliefert, ansonsten wird die IP mit dem niedrigsten numerischen Wert gesetzt.

4.4.2 Alternative Pfade aus demselben Host

Auf einem Host können mehrere Pfade zu einer identischen Datei existieren. Diese verschiedenen Pfade spiegeln sich dann auch in verschiedenen URLs wider. Eine mögliche Ursache für diese Situation besteht in der Verwendung symbolischer Links auf dem Dateisystem des Servers.

4.4.3 Spiegelungen zwischen verschiedenen Hosts

Es können verschiedene Kopien derselben Ressource auf verschiedenen Webservern existieren, beispielsweise die bekannten Mirror Sites. Zudem könnten auch mehrere verschiedene Webserver auf ein gemeinsames geteiltes Dateisystem zugreifen.

Die letzten beiden Fälle werden durch Mercator behandelt, indem zwar die betreffenden Ressourcen heruntergeladen werden, jedoch anschließend durch den Content Seen Test herausgefiltert werden. Die so erzielten Einsparungen sind umso höher, da, würde man eine solches Ressource noch verarbeiten und den Links folgen, so der Overhead im Folgenden stark anwachsen würde.

4.4.4 Sitzungsbezeichner in einer URL

Um das Verhalten von Webseitenbesuchern nachzuverfolgen oder auch um bestimmte Vorgänge, etwa Einkaufsvorgänge, eindeutigen Benutzern zuzuordnen, werden Sitzungsbezeichner bzw. Session IDs in URLs eingefügt. Sie können sich innerhalb einer angehängten Query oder auch innerhalb des Pfads der zurückgelieferten URL befinden. Damit besteht die Möglichkeit, dass eine potenziell unendliche Menge von URLs sich auf dieselbe Ressource beziehen. Es handelt sich hier eigentlich um eine weitere Variante der bereits dargestellten URL Aliase. Der Content Seen Test verhindert in Mercator das weitere Verarbeiten solcher Ressourcen.

4.4.5 Crawlerfallen

Eine Crawlerfalle sind URLs, die zu einem potentiell unendlich langen Crawl führen. Sie können beispielsweise entstehen durch eine zirkuläre Verlinkung von Ressourcen innerhalb eines Dateisystems. Sie können in Kombination mit den Sitzungsbezeichnern in einer URL entstehen oder auch, indem permanent zufälliger Content generiert wird, um den Crawler auf seiner Domain gefangen zu halten. Eine mögliche Motivation besteht hier darin, einen künstlich hohen Page Rank zu erzielen. Eine automatische Detektion in Mercator ist jenseits der bereits erwähnten Methoden nicht implementiert, ein menschlicher Beobachter soll hier stattdessen diese Fallen erkennen und auf entsprechende schwarze Listen setzen. Dies können dann in dem URL Filter berücksichtigt werden.

5 Zur Theorie und Praxis der Clusteranalyse

In diesem Abschnitt soll die Theorie und Praxis von Clusteranalysen innerhalb von informationsverarbeitenden Systemen erläutert werden. Es geht darum zu klären, warum solche Untersuchungen grundsätzlich sinnvoll sind sowie um eine Überschau über verwendete Techniken und ihrer theoretischen Hintergründe.

5.1 Clusteranalyse vs. Klassifikation

Prinzipiell lassen sich mindestens zwei verschiedene Strategien skizzieren, mit denen ein Zusammenhang zwischen den informationstragenden Objekten sowie den Klassen hergestellt werden kann.

5.1.1 Clusteranalyse

Im Rahmen der Clusteranalyse, auch Clustering genannt, werden die zu untersuchenden Objekte genutzt, um erst basierend auf dieser Analyse mögliche Klassen zu identifizieren, denen sie sinnvoll zugeordnet werden können. Es besteht hier kein Vorwissen über bereits existierende Klassen. Die so gewonnen Informationen könnte man beispielsweise zukünftig nutzen, um zukünftige Informationsobjekte diesen Klassen zuordnen zu können.

Die so gewonnen Klassen können in einem folgenden Schritt noch auf ihre Tauglichkeit hinsichtlich des zu untersuchenden Problems analysiert werden, dadurch können eventuelle Korrekturen erfolgen und verschiedene Klassen zusammengeführt oder neue Klassen kreiert werden, die durch das genutzte algorithmische Verfahren nicht identifiziert werden konnten.

5.1.2 Klassifikation

Grundsätzlich versteht man unter dem Begriff der *Klassifikation* eine Operation, nach der Objekte in Klassen bzw. Gruppen eingefügt werden, wobei diese Klassen bzw. Gruppen in der Regel eine geringere Anzahl als die einzufügenden Objekte haben. Der Prozess der Klassifikation kann also verstanden werden als die Zuordnung eines Informationsobjektes zu einer Klasse. Im Gegensatz zur Clusteranalyse ist bei diesem Ansatz bereits im Voraus bekannt, welche Klassen existieren. Dieses Wissen könnten generiert worden sein basierend auf einer vorherigen maschinellen Clusteranalyse oder auch beispielsweise basierend auf bereits durch anderem empirisch gewonnenem Domänenwissen.

5.2 Grundlagen

Die folgenden Abschnitte über grundsätzliche Betrachtungen sowie der historischen Entwicklung rund um das Thema der Clusteranalyse sind sinngemäß der *Internet Encyclopedia of Philosophy* entnommen (vgl. Parrochia o. D.).

Klassifikationsprobleme sind eines der Hauptthemen der wissenschaftlichen Forschung, sie treten beispielsweise auf in der Mathematik, den Naturwissenschaften, den Sozialwissenschaften sowie den Bibliothekswissenschaften (vgl. Hjørland 2017). In ihnen existieren verschiedene Schemata bzw. Verfahren, um Objekte nach bestimmten Kriterien klassifizieren zu können. Diese Schemata nennt man auch *Taxonomien*. Das primäre Ziel der Clusteranalyse sowie Klassifikation besteht in dem Ordnen sowie Organisieren einer inhärent chaotischen und häufig strukturarmen Welt. Durch die Reduktion der Mächtigkeiten der Mengen entstehen Äquivalenzklassen, welche im Folgenden stellvertretend für die sie

umfassenden Objekte untersucht werden können. Durch die damit einhergehende Abstraktion verspricht man sich einen Wissenszuwachs und einen vereinfachten Abruf von Informationen (vgl. Parrochia o. D.).

Meist findet die Klassifikation auf einer endlichen Menge statt, betrachtet man aber auch abstrakte Objekte wie mathematische Strukturen, so sind auch Klassifikationen auf unendlichen Mengen möglich. Der Begriff *Klassifikation* umfasst nicht nur den beschriebenen Vorgang der Zuordnung, ebenfalls kann man das Ergebnis einer solchen Operation als Klassifikation bezeichnen. Eine grundsätzlich anzustrebende Eigenschaft von *Klassifikationsverfahren* besteht in der *Stabilität*, die resultierende Klassifikation soll sich also auch bei kleineren Änderungen der Daten nicht grundlegend verändern. Je nach Art der zu klassifizierenden Objekte und Art des *Klassifikationsverfahrens* können verschiedene Situationen entstehen. Die gebildeten Klassen können disjunkt oder nicht disjunkt sein, sie können in ihrer Anzahl endlich oder auch unendlich sein, sie können die aus der klassischen Mengenlehre bekannte Eigenschaft der *Bestimmtheit* besitzen, nach denen für jedes Objekt eindeutig entscheidbar sein muss, ob es einer Menge angehört oder nicht. Alternativ dazu können die Klassen auch *unscharf* bzw. *fuzzy* definiert sein, so dass ein Objekt einer Klasse auch graduell angehören kann. Die aus einer Clusteranalyse entstehenden Klassen können flach oder hierarchisch angeordnet sein. Kombinationen aus diesen verschiedenen Eigenschaften sowie weitere Situationen sind denkbar.

5.3 Historische Entwicklung

Bereits seit der Antike existieren verschiedene Verfahren für die Klassifikation. Nach der Theorie von Plato sind Klassifikationen als Operation immer endlich, sie kann also nicht unendlich häufig wiederholt ausgeführt werden. Zudem sollen Klassifikationen hierarchisch und dichotom sein. Die Auftrennung basiert in jedem Schritt auf genau einem Kriterium. Sie trennen also mit jedem Schritt eine bestehende Klasse in zwei weitere sich gegenseitig in der Eigenschaft der Auftrennung widersprechende Klassen, die vereint der ursprünglichen Klasse entsprechen. Jedes Objekt gehört in jedem Schritt zu genau einer Klasse. Die Auftrennung erfolgt nach seiner Theorie möglichst symmetrisch, so dass die Mächtigkeiten der entstehenden Klassen ähnlich sind, die Klassifikation also wohlbalanciert ist. Das Kriterium, nach dem die Auftrennung erfolgt, soll dabei inhaltlich sinnvoll sein. Beispielsweise ist eine Auftrennung der natürlichen Zahlen in die zwei Klassen mit den Eigenschaften gerade bzw. nicht gerade oder auch prim bzw. nicht prim inhaltlich sinnvoll. Dagegen wäre eine Auftrennung der natürlichen Zahlen in zwei Klassen, wobei Klasse A alles natürlichen Zahlen <= 1000 enthält und Klasse B die restlichen Zahlen im Allgemeinen weder inhaltlich sinnvoll, noch würden die so entstandenen Klassen wohlbalanciert sein.

Bereits dieses Schema zeigt ein bei vielen Klassifikationsverfahren grundsätzlich bestehendes Problem auf. So soll die Wahl des Trennungskriteriums zwar inhaltlich sinnvoll gestaltet werden, ist aber darüber hinaus beliebig. Daher existieren etwa in der Biologie verschiedene konkurrierende Taxonomien, nach denen Pflanzen beispielsweise einmal anhand ihrer Blütenkronen unterschieden werden, ein anderes Mal anhand der Beschaffenheit ihrer Fortpflanzungsorgane. Beide inhaltlich sinnvoll begründbar, können im Folgenden zu verschiedenen Klassifikationen führen.

Zu Beginn des 19. Jahrhunderts definierte Kant die *logische Division* als die Division von all dem möglicherweise in ihr Enthaltenem. Nach seiner Vorstellung ist eine solche Division disjunkt und ihre Vereinigung stellt die ursprüngliche Klasse wieder her. Dies entspricht bisher auch dem Konzept von Plato. Im Gegensatz zu Plato kann aber jede Klasse immer wiederholt geteilt werden, so dass eine unendliche Klassifikation entsteht.

Weitere bedeutsame Klassifikationsschemata stellen das Periodensystem nach Mendelejew in der Chemie dar, die Dezimalklassifikation nach Dewey sowie die Colon-Klassifikation nach Ranganathan. Die letzten beiden Systeme sind für die Bibliothekswissenschaften relevant, sollen hier aber nicht weiter betrachtet werden.

5.4 Mathematische Ordnungsmodelle

Die folgenden Abschnitte über die mathematische Theorie der Klassifikation sind sinngemäß der *Encyclopedia of Knowledge Organization* entnommen (vgl. Parrochia 2018).

Mathematische Ordnungsmodelle wurden bereits durch Peano, Dedekind und Cantor entwickelt. Später wurde durch Birkhoff mit der Gittertheorie eine bedeutsame Visualisierung für Halbordnungen ermöglicht. Die Bedeutsamkeit von mathematischen Modellen ist durch den Einsatz von Computern weiter angestiegen, nach denen alle möglichen Partitionen, Ketten von Partitionen, Hypergraphen sowie Klassensystemen über einer zu untersuchenden Domäne generiert werden können. Trotz dieses neuen mathematischen Ansatzes basiert ein großer Anteil der Klassifikationen auf der Auswertung von Ähnlichkeiten zwischen Objekten, basierend auf gewonnenen empirischen Daten. Je nach Eigenschaften dieser empirischen Daten bzw. ihrer darauf basierenden Merkmale besitzen diese verschiedene Skalenniveaus. Die Art des Skalenniveaus bestimmt im Weiteren, welche Algorithmen zur Klassifikation mittels welcher mathematischen Operationen anwendbar sind, ebenfalls unterscheiden sich basierend darauf die Arten der Lageparameter. Eine echte Algebra der Klassifikationen, eine Art Metaklassifikation, welche die Eigenschaften sowie die Beziehungen zwischen verschiedenen Klassifikationsverfahren ausdrücken könnte, existiert gegenwärtig noch nicht.

Innerhalb der Wissenschaften setzt eine Klassifikation die Existenz einer Äquivalenzrelation *R* voraus, die zwischen den Elementen einer Menge M definiert ist. Durch diese Äquivalenzrelation entstehen Äquivalenzklassen, diese erzeugen eine Partition auf M. Unter der definierenden Eigenschaft *P* bzw. der Funktion, welche jeweils die Elemente von M ihren Äquivalenzklassen zuweist, verhalten sich die Elemente der Äquivalenzklassen bzgl. P immer identisch, damit stellt die Eigenschaft P eine Invariante dar. Anhand dieser Invarianten können die verschiedenen Elemente verglichen werden. Beispielsweise kann man die Menge $\mathbb{N}$ der natürlichen Zahlen in gerade bzw. ungerade Zahlen einteilen. Als Invariante kann die Funktion $f(x) = x \bmod 2$ angesehen werden, unter dieser Abbildung verhalten sich die Elemente beider Klassen jeweils gleich untereinander und gegenüber den Elementen der jeweils anderen Klasse verschieden. Verallgemeinert kann man beispielsweise verschiedene Mengen klassifizieren basierend auf ihren Mächtigkeiten. In den experimentellen Wissenschaften existieren zudem komplexere Invarianten, beispielsweise die Struktur von Kristallen, welche basierend auf ihrer Symmetrie klassifiziert werden können, wobei die Symmetrie auf der Gruppentheorie basiert. Invarianten stellen damit Kriterien dar, die es erlauben zu entscheiden, ob zu vergleichende Objekte einander ähnlich sind. Allerdings existieren in vielen Fachgebieten keine guten oder unumstrittenen Invarianten zur Klassifikation. In diesen Gebieten kann eine Klassifikation basierend auf dem Konzept der Ähnlichkeit zwischen je zwei Objekten vorgenommen werden. Ähnlichkeit wird hier ausgedrückt in Form eines mathematischen Ähnlichkeitskoeffizienten, welcher wiederrum ein abstraktes Konzept für ein *Distanzmaß* darstellt. Ein solches Distanzmaß setzt seinerseits das Konzept eines *metrischen Raums* voraus. Damit besitzt die Menge M der zu klassifizierenden Elemente eine ihr aufgeprägte Struktur wie beispielsweise eine Ordnung oder ein topologischer Raum.

Im Folgenden werden extensionale Strukturen, also Strukturen, die definiert werden anhand der Elemente, die ihnen jeweils zugeordnet sind, sowie Methoden für den Aufbau von empirischen Klassifikationen näher beleuchtet. Auf den letzteren Methoden basieren auch die bedeutsamsten Algorithmen zur Gewinnung von Klassifikationen.

5.4.1 Extensionale Strukturen

Im Folgenden werden relevante extensionale Strukturen aufgezeigt, beginnend bei den schwächsten Formen, welche sukzessive um weitere Bedingungen erweitert werden, um zu stärkeren Strukturen zu gelangen.

5.4.1.1 Hypergraph

Als schwächste Struktur kann ein Hypergraph angesehen werden. Sei X eine nichtleere endliche Menge und P(X) die Potenzmenge davon. Ein Hypergraph H = (X,K) besteht aus einer Menge von Knoten X sowie einer nichtleeren Menge K, welche aus Teilmengen von X besteht, diese werden auch Kanten genannt. Also gilt: K ∈ P(X)\ Ø.

BSP:

H = ({v_1, v_2, v_3, v_4, v_5}, { {v_3, v_4, v_5}, {v_4, v_5}, {v_2} })

visualisiert:

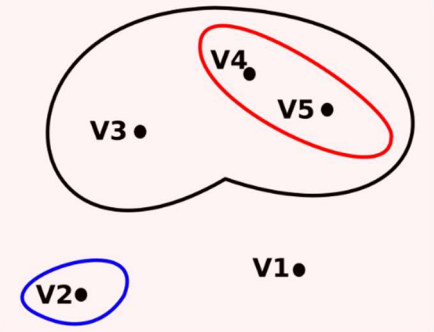

Abbildung 3: Einfacher Hypergraph

Gemäß diesen Bedingungen besteht die Möglichkeit, dass die Menge der Kanten K nicht notwendigerweise die gesamte Menge X überdeckt. Dies impliziert, dass es Knoten geben kann, die keiner Kante zugeordnet sind.

5.4.1.2 Hypergraph mit Überdeckung der Menge

Erweitert man nun den Hypergraph um die folgende Bedingung, nach der die gesamte Menge X auch als Kante existiert, so erhält man als stärkere Struktur eine Menge K als Überdeckung der Menge der Knoten X.

(B1): $X \in K$

5.4.1.3 Klassensystem

Fügt man weiter die Bedingung hinzu, dass jedes Element aus X bzw. jeder Knoten auch eine eigene einelementige Menge (engl. *singleton*) in der Menge der Kanten darstellt, also jeder Knoten auch eine jeweils eigene Kante ist, so gelangt man zu einem Klassensystem (vgl. Brucker/Barthélemy 2007).

(B2): $\forall x \in X: \{x\} \in K$

BSP:

KS = ({v_1, v_2, v_3, v_4, v_5}, { {v_3, v_4, v_5}, {v_4, v_5}, { v_1}, { v_2}, { v_3}, { v_4}, { v_5}, {v_1, v_2, v_3, v_4, v_5})

visualisiert:

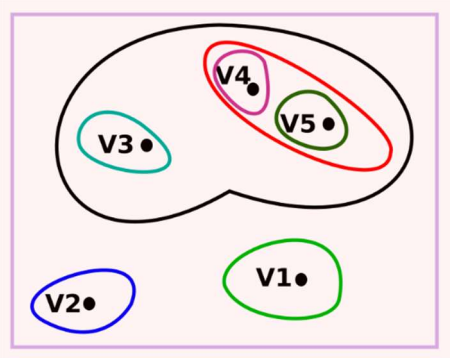

Abbildung 4: Einfaches Klassensystem

5.4.1.4 Partition

Erweitert man nun dieses Klassensystem um die beiden nachfolgenden Bedingungen, so erhält man eine Partition:

(B3): $\forall K_i, K_j \in K: K_i \neq K_j \Rightarrow K_i \cap K_j = \emptyset$

(B4): $K1 \cup K2 \cup K3 \cup \ldots \cup Kn : \bigcup K_i = X$

Die Menge K stellt damit eine Partition von X dar, die einzelnen Elemente K_i nennt man auch Blöcke der Partition.

BSP:

Par = { {v_4, v_5}, { v_1}, { v_2}, { v_3} }

visualisiert:

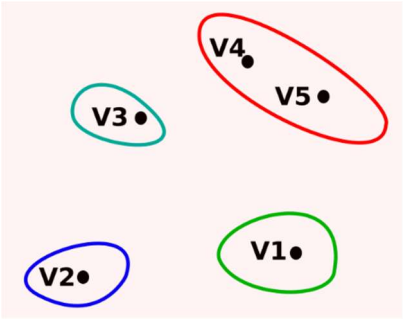

Abbildung 5: Einfache Partition

5.4.1.5 Gitter von Partitionen

Sei nun x R y die Relation „*x gehört zur selben Klasse wie y*“ und bezeichne Par(X) die Menge der Partitionen auf einer nichtleeren endliche Menge X. Dann definiert man eine Beziehung $Par_1 \leq Par_2$ zwischen je zwei Partitionen auf der Menge X. Dabei ist die Partition Par_1 „*als feiner*“ als die Partition Par_2 definiert, wenn gilt: x R y in $Par_1 \Rightarrow$ x R y in Par_2. Die so definierte „*feiner als*“ Beziehung gestattet es, auf der Menge der Partitionen Par(X) eine Halbordnung zu definieren: (Par(X), ≤). Eine solche Halbordnung kann in Form eines Hasse-Diagramms visualisiert werden:

BSP:

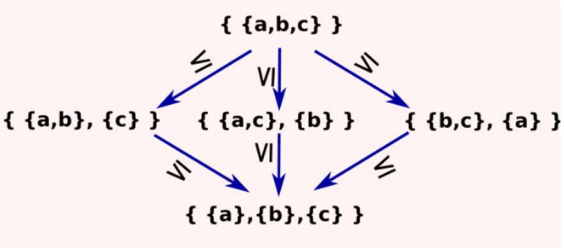

Abbildung 6: Einfaches Gitter von Partitionen (in Anlehnung an Parrochia 2018)

Man kann anhand des Hasse-Diagramms erkennen, dass es ein insgesamt *größtes* bzw. gröbstes Element gibt, die Partition ganz oben im Diagramm, die nur aus einer Menge besteht, die sämtliche Elemente der ihr zugrunde liegenden Menge X enthält. Außerdem gibt es ein insgesamt kleines bzw. feinstes Element, also die feinste Partition ganz unten im Hasse-Diagramm, die Menge die sämtlich aus den einelementigen Mengen der Elemente der ihr zugrunde liegenden Menge X = {a, b, c} besteht. Weiterhin ist wichtig zu erkennen, dass die Beziehung Par ≤ Par' nicht zwischen allen Elementen bzw. Partitionen existiert. So sind die Partitionen { {a, b}, {c} } und { {a, c}, {b} } nicht mittels der „*feiner als*“ Beziehung vergleichbar. Jedoch sind sämtliche Partitionen, die entlang der durch die Pfeile ausgezeichneten Wege existieren mittels dieser Beziehung miteinander vergleichbar, zudem existiert für jedes Paar von Partitionen dort immer ein kleinstes oberes Element bzw. ein größtes unteres Element, damit stellen diese *Partitionsketten* ein Gitter nach Birkhoff dar. Jede der Partitionen aus einer Partitionskette C hat zudem maximal einen Vorgänger sowie maximal einen Nachfolger, zudem stellt jede Partitionskette genau eine mögliche hierarchische Klassifikation dar.

BSP:

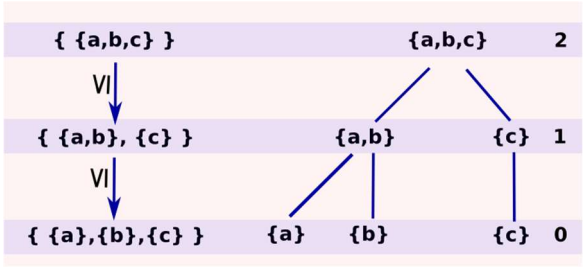

Abbildung 7: Zusammenhang zwischen Partitionskette und hierarchischer Klassifikation (in Anlehnung an Parrochia 2018)

Man kann beweisen, dass sämtlich Partitionsketten eines solchen Gitters äquivalent zu allen möglichen hierarchischen Klassifikationen sind. Damit entspricht die Menge C(X) aller dieser *Partitionsketten* genau der Menge aller möglichen hierarchischen Klassifikationen auf einer Menge und die Menge Par(X) umfasst alle möglichen Partitionen auf X.

Ein Problem bei der Nutzung dieser Methode besteht in der kombinatorischen Explosion von möglichen Partitionen bei linear steigender Anzahl von zu klassifizierenden Objekten. Damit wird die Entscheidung schwierig, welche der theoretisch möglichen Klassifikationen die Beste für eine gegebene Aufgabenstellung darstellt. Ein weiteres noch grundlegenderes Problem besteht darin, dass die Ordnung der Objekte der realen Welt augenscheinlich nicht auf *Partitionsketten* basiert.

5.4.2 Methoden für die Schaffung empirischer Klassifikationen

Während die extensionalen Strukturen rein durch ihre Objekte definiert sind, soll in den meisten Anwendungsfällen eine Klassifikation anhand der Eigenschaften der zu klassifizierenden Objekte vorgenommen werden, welche empirisch gewonnen wurden. Dabei werden ähnliche Objekten zusammengefasst, dies bedingt, dass das Konzept der Ähnlichkeit zwischen je zwei Objekten auf einer Menge präzisiert werden muss.

5.4.2.1 Metrik und Ultrametrik

Falls auf einer Menge von Elementen eine Abbildung bzw. Abstandsfunktion zwischen je zwei Elementen definiert ist, so nennt man diese Abstandsfunktion auch Metrik und die Menge mit dieser aufgeprägten Struktur einen metrischen Raum:

Sei M die Menge mit den zu klassifizierenden Objekten, welche ggf. über gleiche Ausprägungen von Attributen verfügen. Auf dieser Menge sei eine Abbildung *dist* definiert mit M x M → $\mathbb{R}^{0+}$ mit folgenden Eigenschaften:

(E1): dist(x, y) = dist(y, x)
(E2): dist(x, y) = 0 gdw. x=y
(E3): dist(x, z) ≤ dist(x, y) + dist(y, z)
(E4): dist(x, z) ≤ max{dist(x, y), dist(y, z)}

Falls die Abstandsfunktion die ersten drei Eigenschaften erfüllt für jedes Wertepaar (x,y) ∈ M x M, also jedes Paar von Objekten aus M, so spricht man von einer Metrik. Falls zusätzlich die Eigenschaft E4 erfüllt wird, handelt es sich um eine Ultrametrik. Beispielsweise kann der dreidimensionale Raum unserer Anschauung in Form eines kartesischen Koordinatensystems dargestellt werden, er wird auch *euklidischer Raum* genannt. Dieser euklidische Raum ist ein Spezialfall eines metrischen Raums. Insbesondere besagt die Eigenschaft E3 hier, dass der kürzeste Weg zwischen zwei Punkten immer eine Gerade ist (Dreiecksungleichung). Allerdings handelt es sich nicht um einen ultrametrischen Raum, da geometrische Objekte wie beispielsweise unregelmäßige Dreiecke in unserem euklidischen Raum existieren, welche die Eigenschaft E4 verletzen. Relevant für das Thema der Klassifikation ist, dass ein mathematischer Beweis dafür existiert, nach dem immer eine ganzzahlige Ultrametrik für jede Partitionskette existiert. Eine Ultrametrik kann ausgedrückt werden mittels einer Distanzmatrix. Dabei kann man als Abstand zwischen zwei Objekten auf einem Baum die erste Ebene definieren, in der sich die Objekte in derselbe Klasse befinden.

BSP:

Beispielsweise erhält man aus der Partitionskette, die in Abbildung 7 dargestellt ist, für die damit korrespondiere Darstellung als Baum mit 3 Ebenen die folgende Distanzmatrix zwischen den einzelnen Objekten:

	a	b	c
a	0	1	2
b	1	0	2
c	2	2	0

Tabelle 1: Distanzmatrix

5.4.2.2 Methoden zur Gewinnung einer hierarchischen Klassifikation

Prinzipiell kann man nach den Strategien Bottom-Up oder Top-Down für die Gewinnung eines hierarchischen Klassifikationsschemas. Unabhängig von der konkret genutzten Methode bzw. dem konkret genutzten Algorithmus sind als vorbereitende Maßnahme zunächst sämtliche zu klassifizierende Objekte abzugrenzen. Anschließend werden anhand dieser Objekte die Attribute gewählt, die für die Definition einer Abstandsfunktion benötigt werden. Die Wahl der Attribute sollte, um sinnhaft zu sein, mit dem durch die Klassifikation zu lösendem Problem in Beziehung stehen, falls eine solche Beziehung existiert und bekannt ist. Zudem bestimmt die Wahl der Attribute aufgrund ihrer verschiedenen Skalenniveaus die später anwendbaren Methoden mit. In einem weiteren Schritt wird eine Abstandsfunktion zwischen den Klassen definiert. Für den Sonderfall von einelementigen Klassen muss sie der Abstandsfunktion zwischen den Objekten entsprechen.

5.4.2.2.1Bottom-Up Methoden

Es existieren verschiedene Methoden zur Gewinnung einer Hierarchie dargestellt als Baumstruktur von den Blättern an, also den Klassen jeweils aus einem Objekt bestehend, hin zur Wurzel. Diese Verfahren nennt man auch *agglomerative Clusterverfahren*. Falls die Gesamtanzahl der zu klassifizierenden Objekte ein Vielfaches von 2 ist, können binäre Ketten gebildet werden. Beispielsweise kann die erste Partition dadurch gebildet werden, dass man die zwei Objekte mit dem geringsten Abstand zueinander zu einer Klasse zusammenfasst, die restlichen Elemente bilden dann die zu ihr komplementäre Klasse und diese beiden Klassen stellen wiederum eine Partition auf einer Baumebene dar. Die Zusammenfassung je zweier Teilmengen kann dann innerhalb des binären Baums als Knoten dargestellt werden. Mittels einer optimierenden Zielfunktion kann die Partition gewählt werden, für die die Funktion ein Maximum erreicht. Es existieren verschiedenste Abstandsfunktionen, häufig findet jedoch in Anlehnung an die Physik eine Interpretation der zu klassifizierenden Elemente als eine Punktwolke dar, gelegen in einem meist euklidischen Raum. Für Teilwolken dieser Punktwolke kann anschließend ein Schwerpunkt berechnet werden sowie ein Maß für die Varianz bzw. die Kompaktheit der jeweiligen Teilwolken. Man kann anschließend nach Punktepaaren suchen, für die die Kompaktheit steigt bzw. die Varianz abnimmt. Für diese Suche existieren verschiedene Strategien, nach denen die darauf basierenden Algorithmen benannt sind.

5.4.2.2.2Top-Down Methoden

Analog kann man auch beginnend mit der Gesamtheit der zu klassifizierenden Objekte als Partition eine Folge von Auftrennungen vornehmen. Diese Verfahren nennt man auch *divisive Clusterverfahren*. Je nach Art des zu kreierenden Baums in jeweils zwei oder auch mehr verschiedene Klassen je Baumebene. Beispielsweise kann die Auftrennung erfolgen anhand des maximalen Durchmessers, falls sich die Objekte als Punktwolken in einem euklidischen Raum interpretieren lassen. Eine andere Strategie besteht darin, die Klasse mit der maximalen Anzahl von Objekten aufzutrennen.

5.4.2.3 Methoden zur Gewinnung einer nicht-hierarchischen Klassifikation

Die folgenden Abschnitte über die Theorie der nicht-hierarchischen bzw. flachen Gewinnung von Klassen basieren auf dem Werk *Knowledge Discovery in Databases* (vgl. Ester 2013: 51–76).

Diese Methoden werden auch häufig *partitionierende Verfahren* genannt, da die aus ihnen hervorgehenden Klassen i. d. R. die Eigenschaften einer Partition aufweisen. Abweichend davon existieren noch Verfahren, die die Zugehörigkeit eines Objekts zu einer Klasse mit einer Wahrscheinlichkeit zwischen 0.0 und 1.0 ausweisen. Hier könnten ggf. nichtleere Schnittmengen existieren, falls Objekte mit einer Wahrscheinlichkeit > 0.0 mehreren Klassen angehört. So werden beispielsweise bei dem *EM-Algorithmus* (vgl. Dempster et al. 1977), die Cluster durch eine Wahrscheinlichkeitsverteilung beschreiben. Man kann jedoch i. d. R. die Unschärfe beseitigen, indem man ein Objekt der Klasse vollständig zuweist, zu der es mit der höchsten Wahrscheinlichkeit zugehörig ist. Im Gegensatz zu diesen Methoden haben die beiden hier im Folgenden kurz umrissenen Methoden gemeinsam, dass ein Cluster jeweils durch einen repräsentativen Punkt beschrieben wird.

5.4.2.3.1Clustering durch Minimierung der Varianz

Dieses Verfahren von Forgy (vgl. Forgy 1965) bedingt, dass die Merkmale der zu untersuchenden Objekte metrisch skaliert sind. Die Punkte befinden sich dann in einem euklidischen Vektorraum, dessen Dimension der Anzahl der Merkmale entspricht, man nennt ihn auch *Featureraum*. Daraus ergibt sich, dass zwischen je zwei Punkten die euklidische Distanz existiert. Jede der k Teilwolken als Klassen dieser Punktwolke kann dann durch ihren *Zentroid* repräsentiert werden, dieser entspricht dem elementweisen arithmetischen Mittel aller Punkte innerhalb seiner Teilwolke und kann also der geometrische Schwerpunkt interpretiert werden. Das Ziel besteht nun darin, diese k Teilwolken möglichst kompakt zu gestalten. also die Abstände der Punkte je Teilwolke zu ihren Zentroiden zu minimieren bzw. die Varianz zu ihnen zu minimieren, um damit eine optimale Zerlegung der Punktwolke in Cluster zu erreichen. Der Algorithmus von Forgy beginnt mit einer zufälligen Zuordnung der Objekte bzw. ihrer Repräsentation als Punkte zu einer der k Klassen. Aus diesen Clustern wird dann der Zentroid berechnet. Im nächsten Schritt findet dann ggf. eine neue Zuordnung der Punkte zu den Clustern mit den Zentroiden statt, zu denen sie eine geringere Distanz aufweisen. Basierend auf dem Zustand nach diesem Schritt werden wieder die ggf. neuen Zentroide dieser Cluster berechnet. Diese letzten beiden Schritte der neuen Zuordnung zu anderen Clustern sowie der Neuberechnung der ggf. veränderten Zentroide wird so lange durchiteriert, bis keine Veränderungen in diesen Schritten mehr stattfinden. Damit konvergiert der Algorithmus gegen ein Minimum, welches aber auch noch lokal sein kann. Das finale Ergebnis kann sich daher bezogen auf eine identische Ausgangsbasis von Objekten bei unterschiedlicher initialer Zerlegung unterscheiden.

5.4.2.3.2k-means Methode

Dieses Verfahren von MacQueen (vgl. MacQueen 1967) entspricht im Wesentlichen dem Clustering durch Minimierung der Varianz. Die Neuzuordnung der Cluster sowie die darauf basierende Neuberechnung der Zentroide findet jedoch hier in einem Schritt statt. Es existieren verschiedene Heuristiken zur Auffindung einer bereits sinnvollen initialen Zerlegung sowie Methoden, in denen k-means um Elemente wie das Verschmelzen oder Auftrennen von Clustern erweitert wird, um eine bessere Optimierung zu ermöglichen. Sowohl k-means als auch das Clustering durch Varianzminimierung sind gegenüber Ausreißern empfindlich, dies liegt begründet in der Empfindlichkeit des Arithmetischen Mittels gegenüber Ausreißern.

5.4.2.3.3k-medoid Methoden

Als Vorbedingung zur Anwendung der k-medoid Methoden von Kaufmann und Rousseeuw (vgl. Kaufman/Rousseeuw 1990) sowie der darauf aufsetzenden Verfeinerungen benötigt man lediglich Objekte beliebiger zwischen denen eine ebenfalls beliebige Distanzfunktion oder Distanzmatrix definiert ist. Nach diesem Verfahren wird jeder Cluster repräsentiert durch sein jeweils zentralstes Element. Ein Medoid ist also per Definition auch immer ein Objekt der zu untersuchenden Menge und wird nicht wie beispielsweise die Zentroide errechnet. Falls nun k Medoide existieren, so entspricht das darauf basierende Clustering einer Aufteilung in k Cluster, in denen dann jedes Objekt dem Cluster zugeordnet ist, zu dessen Medoid es den geringsten Abstand aufweist. Das Ziel der Methoden besteht darin, die Wahl der k Medoide so zu treffen, dass die Summe aller Abstände der Objekte zu ihren jeweiligen Medoiden insgesamt minimiert wird, die jeweiligen Cluster also möglichst kompakt sind. Der Algorithmus *PAM* (=Partitioning Around Medoids) von Kaufmann und Rousseeuw beginnt damit, das insgesamt zentralste Objekt als initialen Medoiden zu ermitteln. Nachfolgend werden die nächsten k-1 Medoide ermittelt, bei denen jeweils unter Berücksichtigung der bereits ermittelten Medoide die gesamte Kompaktheit möglichst minimal ist. Anschließend wird der Wert für die Kompaktheit des gesamten Clusterings ermittelt für jede mögliche Vertauschung je eines bisherigen Medoids mit einem anderen Objekt aus der Menge der Daten exklusive der bereits gesetzten Medoide und die so entstandene gesamte Kompaktheit ermittelt. Falls diese Konfiguration kompakter ausfällt, wird die Vertauschung vorgenommen. Dieser Vorgang wird so lange wiederholt, bis das Maß für die Kompaktheit des gesamten Clusterings nicht weiter reduziert werden kann, jeder der k Cluster also möglichst kompakt ist. Obwohl PAM eine sehr umfangreiche und algorithmisch komplexe Suche vornimmt, konvergiert er gegen ein möglicherweise nur lokales Minimum. Alternative Algorithmen wie beispielsweise *CLARANS* (=Clustering Large Applications based on RANdomized Search) von *Ng und Han* (vgl. Ng/Han 1994) arbeiten mit einer weniger ausführlichen Suche und besitzen damit eine geringere algorithmische Komplexität. Sie konnten aufzeigen, dass ihr Algorithmus in der Qualität bzw. erzielten insgesamten Optimierung im Ergebnis gegenüber PAM nur relativ geringe Einbußen aufweist, jedoch ein Laufzeitvorteil von beinahe einer Zehnerpotenz besitzt, je nach Umfang der zu clusternden Objekte.

5.4.2.3.4Treffen einer geeigneten Wahl für die Initialisierung und Anzahl k der Cluster

Die drei obigen vorgestellten Clusteringmethoden basieren alle auf einer ähnlichen Vorgehensweise. In einem ersten Schritt werden jeweils k Cluster erzeugt. Dies kann passieren, indem Medoide gewählt werden oder indem die zu clusternden Objekte in k Gruppen aufgeteilt werden, um aus ihnen jeweils Zentroide zu errechnen. Danach wird die Clusterkonfiguration so lange iterativ optimiert, bis keine Verbesserung mehr erzielt werden kann. Eine relevante Eigenschaft der bisherigen Methoden ist, dass sie jeweils i.d.R. nur lokal optimieren. Daher bestimmt eine bereits sinnvoll getroffene erste Initialisierung maßgebend die Wahrscheinlichkeit, nach Ablauf der Methode gegen ein globales Optimum zu konvergieren. Des Weiteren ist den bisherigen Methoden gemein, dass immer bereits im Vorhinein die Anzahl k der Cluster angegeben werden muss. Im Folgenden soll daher auch umrissen werden, mittels welcher Heuristiken diese Anzahl k möglichst gut bestimmt werden kann.

Falls man mittels geeigneter Heuristiken die initiale Clusteraufteilung bereits nahe an einer angenommenen wahren Clusterstruktur annähern kann, so werden die Ergebnisse der Algorithmen eine insgesamt höhere Qualität aufweisen bzw. mit höherer Wahrscheinlichkeit gegen ein globales Optimum konvergieren. Außerdem wird die Laufzeit der Algorithmen dadurch profitieren, da bereits nach einer geringeren Anzahl von Iterationen keine weitere Verbesserung mehr stattfinden wird.

Es existieren verschiedene Heuristiken für eine sinnvolles initiales Clustering. Die Heuristik von *Fayyad et al.* (vgl. Fayyad et al. 1998) beruht auf dem Prinzip, dass bei wiederholtem Ziehe von Stichproben aus der Menge der zu clusternden Objekte die Lage hin zu den wahren Clusterzentren tendieren. Dies liegt begründet in der höheren Wahrscheinlichkeit, Objekte aus diesen dichter besiedelten Bereichen zu treffen. Nachteilig bei

diesem Verfahren kann sein, dass auch zufälligerweise nicht gute Stichproben gezogen werden. Um diese negativen Effekte einzelner „Ausreißer“-Stichproben zu beheben, zieht man m Stichproben unabhängig voneinander. Je gezogener Stichprobe wendet man auf sie den jeweiligen Clusteringalgorithmus an und erhält damit je Stichprobe eine Schätzung für die k möglichen Cluster bzw. ihrer Mittelpunkte. Gemäß der obigen Grundannahme sind davon bereits mit einer recht hohen Wahrscheinlichkeit ein großer Teil der ermittelten Cluster nahe an den tatsächlichen Zentren gelegen. Anschließend fasst man die k × m Ergebnisse der Stichproben zu einer Menge zusammen. Die Menge dieser Objekte clustert man dann jeweils mit dem zuvor gewonnenen Ergebnis einer Stichprobe als neue initiale Ausgangsbasis. So erhält man final wieder m Ergebnisse mit je k Clusterzentren. Unter diesen Ergebnissen wählt man das Ergebnis, für welches man die beste Kompaktheit bzw. Optimierung erzeugt hat und wendet dieses als Ausgangsbasis für das Clustering aller Objekte an. So wird der Einfluss von Ausreißern unter den Stichproben gedämpft.

Alle bisherigen Algorithmen verlangen im Voraus die Anzahl k der Cluster als Eingaben. In der Realität ist jedoch die Anzahl der Cluster nicht immer im Voraus bekannt. Da ein insgesamt gutes Clustering aber auch von dieser Anzahl abhängt, ist es sinnvoll, diese im Voraus so gut wie möglich abschätzen zu können. Zum einen kann man je nach Lage der Punkte bereits visuell die Anzahl der Zentren bestimmen, insbesondere wenn es sich um *Gaußcluster* handelt, also Punktwolken in Form von Kreisen bzw. Kugeln, bei denen die Dichte hin zum Zentrum zunimmt. Es existieren darüber hinaus aber ebenfalls rein algorithmische Verfahren. Nach diesen führt man zunächst jeweiligen Clusteralgorithmus aus für sämtliche Werte für k von 2 bis -1. Unter den so gewonnen Ergebnissen ist nun das optimale Clustering auszuwählen. Schwierig hierbei ist es, ein Gütemaß für die jeweiligen Clusterkonfigurationen zu finden, welches selbst unabhängig von der Anzahl k der Cluster ist. Die bisher genutzten Gütemaße für eine Clusterkonfiguration basieren jeweils auf der Kompaktheit des Clusterings, dieses hängt jedoch direkt von der Anzahl k der Cluster ab. Mit ihm wird das Clustering mit steigendem k immer besser, im Grenzfall wird damit das beste Clustering erreicht, indem jedes Objekt einen eigenen Cluster darstellt. Ein Maß, welches diesen Nachteil nicht besitzt, ist die Silhouette, mit dem auf ihr basierenden Silhouettenkoeffizient kann ermittelt werden, welche Güte ein Clustering besitzt. Vereinfacht ausgedrückt gibt die Silhouette für ein Objekt, welches einem Cluster zugewiesen ist, an, wie gut die Zuordnung zu diesem Cluster ist. Um dieses zu ermitteln wird der durchschnittliche Abstand des Objekts zu allen anderen Objekten seines Clusters ermittelt und mit dem durchschnittlichen Abstand des Objektes zu allen anderen Objekten seines ihm nächstgelegenen anderen Clusters, ermittelt. Der daraus gewonnene normierte Koeffizient erlaubt dann eine Aussage darüber, ob die aktuelle Zuordnung des Objektes sinnvoll ist, es besser dem anderen nächstgelegenen Cluster zuzuordnen wäre, oder der Zustand indifferent ist. Indifferenz kann man dahingehend interpretieren, dass das Objekt einen Grenzfall darstellt. Ermittelt man nun für jedes Objekt der Gesamtmenge seine Silhouette und bildet das arithmetische Mittel der so gewonnen Summe von Silhouetten, so gelangt man zum Silhouettenkoeffizient, als ein Gütemaß für eine gesamte Clusterkonfiguration.

5.4.2.3.5Grenzen der bisherigen Methoden

Die bisher erläuterten Methoden arbeiten nicht in jedem Fall zielführend. Beispielsweise implizieren diese Methoden, dass die Cluster eine kreis- bzw. kugelförmige Struktur besitzen. Tatsächlich gibt es jedoch darüber hinaus weitere mögliche Formen, beispielsweise könnte der „wahre“ Cluster sternförmig sein. Ebenfalls muss er nicht die gesamte Fläche umfassen, die er nach außen hin begrenzt, er könnte in zweidimensionaler Form beispielsweise einem Kreisring entsprechen mit einer Aussparung im Inneren. Weitere Probleme können entstehen, wenn die Cluster sich in ihrer Ausdehnung stark voneinander unterscheiden, sowie, wenn sie eine sehr unterschiedliche Punktedichte aufweisen. In diesen Fällen sind die durch die Algorithmen generierten Cluster nicht mehr gut nutzbar, zudem kann in diesen Konstellationen auch der Silhouettenkoeffizient versagen.

Lösungen für diese Probleme können in Ansätzen bestehen, in denen die Dichte von Punktmengen berechnet werden kann, angelehnt an Methoden der Analysis wie beispielsweise der Epsilon-Umgebung.

5.4.3 Offene Probleme empirisch basierter Klassifikationen

Obwohl wie bereits dargestellt, verschiedenste Ansätze bestehen, ein hierarchisches oder auch nicht-hierarchisches Clustering durchzuführen, bleibt das grundsätzliche Problem der Instabilität bestehen. Die Instabilität der gewonnenen Klassifikationen beruht dabei auf einer *intrinsischen Instabilität* sowie einer *extrinsischen Instabilität*.

Die intrinsische Instabilität rührt daher, dass es verschiedene Formeln für die Berechnung von Distanzfunktionen gibt, ebenfalls können die zu entdeckenden Klassen mittels verschiedener Verfahren bzw. Algorithmen gewonnen werden. Variiert man einen oder beide dieser Faktoren, so erhält man im Ergebnis verschiedene Klassifikationen. Falls man eine nur kleine Menge von Objekten betrachtet, welche hierarchisch angeordnet werden können, so muss eine der möglichen Partitionsketten die wahre hierarchische Klassifikation darstellen, für diese existiert dann auch eine eindeutige Ultrametrik (vgl. Kapitel 8.4.2.1). Man könnte dann für jede existierende Definition eines Distanzfunktion zwischen diesen Objekten prüfen, inwiefern diese eine Ultrametrik approximiert. Falls nun eine Distanzfunktion einer Ultrametrik nahekommt, so korrespondiert diese mit hoher Wahrscheinlichkeit mit einer eindeutigen hierarchischen Klassifikation. Jedoch existieren verschiedenste theoretisch mögliche hierarchische Klassifikationen und welche davon der realen Gegebenheiten am nächsten kommt ist nicht ohne besonderes Vorwissen bekannt. Besäße man Vorwissen jedoch, so müsste man keine Clusteranalyse mehr betreiben, sondern sich nur noch um die Klassifikation kümmern, also der Zuordnung der Objekte zu einem bereits bestehenden Klassifikationsschema (vgl. Parrochia 2018).

Die extrinsische Instabilität basiert auf dem sich verändernden Wissen. Diese kann beispielsweise daher entstehen, da sich das Wissen über die relevanten Eigenschaften von Objekten im Laufe der Zeit ändert und damit beispielsweise auch grundlegende Definitionen angepasst werden müssen (vgl. Parrochia 2018).

In dem speziellen Betrachtungsfall dieser Arbeit basierend die relevanten Metadaten auf klar definierten Eigenschaften, daher sollte das Problem der extrinsischen Instabilität für das Clustering nicht besonders relevant sein.

5.4.4 Kleinbergs Unmöglichkeitstheorem für das Clustern

Jon Kleinberg hat in einer Forschungsarbeit im Jahr 2002 geprüft, ob zwischen den verschiedenen bisherigen Clustertechniken für eine Menge von verschiedenen und für das Clustering prinzipiell wichtigen Eigenschaften in ihrer Erfüllung Zielkonflikte bestehen können. Während die grundlegende Idee des Clusterings von Punkten intuitiv einfach verständlich ist, existieren in der konkreten Implementierung verschiedene Methodiken, von denen eine Auswahl bereits in der Arbeit dargestellt wurden. Einem großen Teil dieser bisherigen Betrachtungsweisen ist gemein, dass man das Clustering immer in Bezug auf eine oder mehrere bestimmte Methodiken untersucht hat, jedoch selten unabhängig davon. Kleinbergs Herangehensweise besteht abweichend davon darin, dass er formaltheoretisch ein *axiomatisches Gerüst* aufstellt. In diesem führt er die für das Ziel des Clusterings wichtigen Eigenschaften ein und prüft anschließend, inwiefern diese Eigenschaften die Lösungen, die man gewinnen kann, beschränken. Als Grundlage seiner Untersuchungen dient eine *Clusterfunktion f()*, die eine vorgegebene Menge von zu clusternden Objekten sowie die jeweils paarweise Distanzen zwischen ihnen erhält. Sie liefert eine Partition Γ auf dieser Menge zurück. Diese minimale Vorbedingung impliziert, dass die aus ihr gewonnenen Betrachtungen auf beliebige Punkte in einer Menge

verallgemeinert werden können, diese müssen dabei nicht in einem metrischen Raum liegen, womit die Betrachtung beispielsweise auch für das Clustering von kategorialen Attributen relevant ist.

Anschließend werden die folgenden drei Eigenschaften betrachtet, die eine Clusterfunktion aufweisen sollte:

- **Skaleninvarianz bzw. Skalenunabhängigkeit**: Diese besagt, dass falls die Skalen der zu betrachtenden Eigenschaften transformiert werde, so dass beispielsweise die Datenpunkte durch diese Transformation weiter auseinander liegen, dann diese Transformation das Ergebnis des Clusterings nicht verändern darf, da die Transformation für alle Punkte gleichartig ist.

  Formal kann dies dargestellt werden, indem man eine Distanzfunktion dist() annimmt. α×dist() bezeichnet dann die Distanzfunktion, für die die Distanz zwischen i und j α×dist(i,j) entspricht. Nun gilt für jede Clusterfunktion f() und jede Distanzfunktion dist() mit jedem α > 0: f(dist()) = f(α×dist(i,j))

  Dies ist zu unterscheiden von den verschiedenen Skalenniveaus im statistischen Sinne, wie in der Einleitung kurz dargestellt.

- **Reichhaltigkeit**: Nach dieser Eigenschaft sollte die Clusterfunktion theoretisch in der Lage sein, sämtliche mögliche Partitionen auf der eingegebenen Menge als Output zu erzeugen. Dies ist notwendig, da theoretische auch beliebige Partitionen als Ergebnis auftreten können müssen, man also den Raum der Lösungen nicht bereit im Vorhinein begrenzen darf.

  Formal kann dies dargestellt werden, indem man mit dem Wertebereich einer beliebigen Clusterfunktion f() auf der Grundmenge der zu clusternden Objekte die Menge aller möglichen Partitionen Γ bezeichnet, so dass f(dist()) = Γ für eine Distanzfunktion dist() gilt.

- **Konsistenz**: Diese Eigenschaft besagt, dass falls die Daten individual so transformiert werden, dass die Abstände der Punkte innerhalb eines Clusters herabgesetzt werden und die Abstände der Punkte zwischen verschiedenen Clustern heraufgesetzt werden, dann das Ergebnis des Clusterings sich dadurch nicht verändern sollte, da die wesentliche Struktur erhalten bleibt.

  Formal kann dies dargestellt werden, indem man annimmt, dass Γ eine Partition auf der Grundmenge der zu clusternden Objekten darstellt. Seien zudem dist() sowie $dist()^I$ zwei Distanzfunktionen auf der Grundmenge. Man bezeichnet dann $dist()^I$ eine Γ-Transformation der Funktion dist(), falls gilt:

  (i) für alle i,j ∈ Grundmenge M die zu demselben Cluster von Γ gehören gilt, dass $dist(i, j)^I \leq dist(i, j)$ sowie
  (ii) für alle i,j ∈ Grundmenge M die zu anderen Clustern von Γ gehören gilt, dass $dist(i, j)^I \geq dist(i, j)$.

  Seien nun dist() sowie $dist()^I$ zwei Distanzfunktionen. Falls für eine Clusterfunktion f() gilt: f(dist()) = Γ und $dist()^I$ eine Γ-Transformation von dist() dargestellt, dann gilt: $f(dist()^I) = Γ$.
  Falls also ein Clustering Γ durch eine Distanzfunktion dist() entsteht und man nun die neue Distanzfunktion $dist()^I$ anwendet, indem die Abstände der Punkte wie beschrieben verändert wird, man dasselbe Clusterergebnis Γ erhält.

Die Kernaussage seiner Untersuchungen lautet, dass es keine Clusterfunktion geben kann, welche alle drei Eigenschaften zugleich besitzt. Zudem konnte aufgezeigt werden, dass auch keine dieser drei Eigenschaften redundant ist. Man kann jeweils leicht Clusterfunktionen finden, welche je zwei der drei Eigenschaften erfüllen, zwischen den Mengen der erfüllten Eigenschaften und der jeweils fehlenden Eigenschaft besteht dann ein Zielkonflikt. Ebenfalls konnte dargestellt werden, dass wenn man einige dieser drei Bedingungen lockert, man zu einer Menge von Bedingungen gelangt, welche durch bereits bestehende Methodiken wie etwa durch

Single-Linkage erfüllt ist. Ebenfalls kann durch diese Untersuchungen die Menge aller möglichen Outputs einer Clusterfunktion dargestellt werden, falls diese die Eigenschaften der Skaleninvarianz sowie Konsistenz erfüllt. Die Erkenntnisse stellen die in der Aufgabe des Clusterns inhärenten Zielkonflikte dar und beleuchten die verschiedenen Clusteringtechniken nicht bezüglich der Definition der Distanz sowie der genutzten Algorithmen, sondern aus der Warte der Wahl, die bezüglich dieser impliziten Zielkonflikte jeweils getroffen werden (vgl. Kleinberg 2002).

Da das Theorem von Kleinberg eine so fundamentale Auswirkung auf sämtliche Aspekte des Clusterings besitzt, soll noch auf einen weiteren relevanten Zielkonflikt im Kontext seines Theorems eingegangen werde. (vgl. Clustering -- Intuition behind Kleinberg's Impossibility Theorem 2015).

Demnach stützt sich jeder der Clusteralgorithmen auf ein Konzept von Nähe bzw. Distanz ab. Man kann nun entweder ein relatives Konzept von Nähe nutzen, welches mit der Eigenschaft der Skalenunabhängigkeit korrespondiert, oder aber ein absolutes Konzept von Nähe, welches mit der Eigenschaft der Konsistenz korrespondiert. Die gleichzeitige Anwendung beider Konzepte dagegen ist problematisch.

BSP:

Man nehme beispielsweise an, dass zwei zu clusternde Mengen M1 und M2 eine Anzahl von jeweils 220 Punkten besitzen. Sie seien räumlich wie folgt angeordnet:

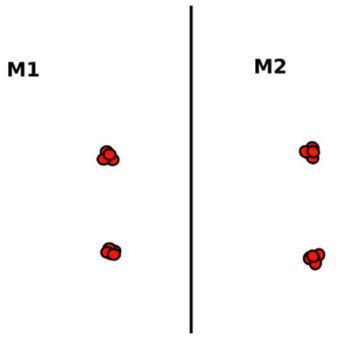

Abbildung 8: Zwei Mengen zu clusternder Objekte (in Anlehung an Clustering -- Intuition behind Kleinberg's Impossibility Theorem 2015)

Aufgrund der „Auflösung" bzw. aktuellen Skala erkennt man jeweils nicht alle Punkte einzeln, sondern nur je Menge drei Anhäufungen, da die Punkte aus aktueller Sicht nahe beieinander liegen. „Zoomt" man nun jedoch in eine der Punktanhäufungen aus der Menge M1 hinein, ändert also die verwendete Skala, so ergibt sich das folgende Bild:

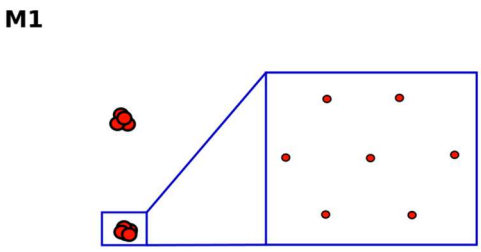

Abbildung 9: Punkteverteilung bei Skalenänderung (in Anlehnung an Clustering -- Intuition behind Kleinberg's Impossibility Theorem 2015)

Basierend auf dieser Erkenntnis könnte man nun verallgemeinert annehmen, dass die zu clusternden Punkte sowohl in den Menge M1 als auch in M2 jeweils in zwei Clustern angeordnet sind. Vergrößert man dagegen auch eine der beiden Anhäufungen der Menge M2, ergibt sich folgendes Bild:

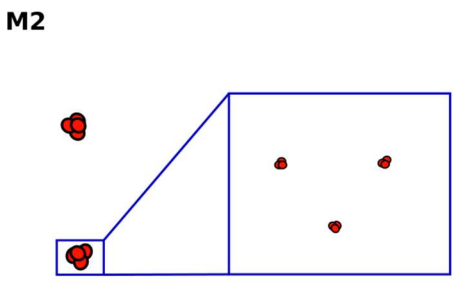

Abbildung 10: Punkteverteilung bei Skalenänderung (in Anlehnung an Clustering -- Intuition behind Kleinberg's Impossibility Theorem 2015)

Vertritt man nun die vorherrschende Ansicht, dass ein absolutes Konzept von Nähe, welches mit der Eigenschaft der Konsistenz korrespondiert, existiert, wird man weiterhin davon ausgehen, dass je Menge M1 und M2 zwei verschiedene Cluster existieren. Der einzige Unterschied zwischen M1 und M2 besteht darin, dass innerhalb der Anhäufungen der Punkte diese in M2 wiederum sich um drei Zentren gruppieren und dort enger beieinander liegen. Falls dagegen in der individuellen Betrachtung ein relatives Konzept von Nähe präferiert wird, welches mit der Eigenschaft der Skalenunabhängigkeit korrespondiert, kann man dagegen argumentieren, dass innerhalb der Menge M2 2 x 3 = 6 Cluster existieren. Beide Betrachtungsweisen sind valide, hängen aber von der Betrachtungsweise ab und führen in diesem Fall zu verschiedenen Ergebnissen.

Gemäß Kleinbergs Theorem ist es jedoch möglich, die beiden Eigenschaften der Skalenunabhängigkeit und Konsistenz gemeinsam zu bewahren. Dies ist möglich, wenn man auf die Eigenschaft der Reichhaltigkeit verzichtet. Jedoch verliert man bei der Beibehaltung von Skalenunabhängigkeit und Konsistenz eine weitere Eigenschaft, die sogenannte *Isometrie-Invarianz*. Falls ein Algorithmus die Eigenschaft der Isometrie-Invarianz erfüllt, so hängt das von ihm gewonnene Ergebnis ausschließlich von den Abständen zwischen den Punkten ab. Keine weiteren Eigenschaften wie beispielsweise eine Ordnung bzw. Reihenfolge oder auch einfache Namen sind zusätzlich für die Punkte nötig. Wenn man sich die zu clusternden Punkte in einem metrischen Raum vorstellt, so besagt die Eigenschaft der Isometrie-Invarianz, dass ein Clusteringalgorithmus gegenüber Rotationen, Reflektionen sowie Translationen unempfindlich ist, also bei identischem Input jeweils auch einen identischen Output erzeugt. Neben der Reichhaltigkeit würde man diese essentielle Eigenschaft verlieren, falls ein Clusteringalgorithmus zugleich skalenunabhängig und konsistent ist

6 WWW als Graph betrachtet

Das WWW kann man als einen gerichteten Graphen auffassen, wobei die Knoten Ressourcen sind (beispielsweise HTML-Seiten oder auch PDF-Dateien) und die gerichteten Kanten als Links aufgefasst werden können. Innerhalb dieses Graphen kann es auch zwischen je zwei Knoten mehrere Kanten geben. So könnte beispielsweise die Ressource *A* an mehreren Stellen auf eine Ressource *B* mittels Hyperlinks verweisen. In dieser Abhandlung wird diese *Vielfachheit* der möglichen Verknüpfungen zwischen zwei Ressourcen A und B nicht weiter berücksichtigt, sondern im Weiteren abstrahiert zu genau einer gerichteten Kante von *A* nach *B*. Ebenfalls können sich die Ressourcen *A* und *B* auch jeweils gegenseitig verlinken, müssen dies jedoch nicht. Ein weiterer Fall ist ein selbstreferentieller Verweis, eine Ressource verweist also auf sich selbst.

In der folgenden Graphik ist ein exemplarischer Webgraph abgebildet, ohne Berücksichtigung seiner weiteren Umgebung:

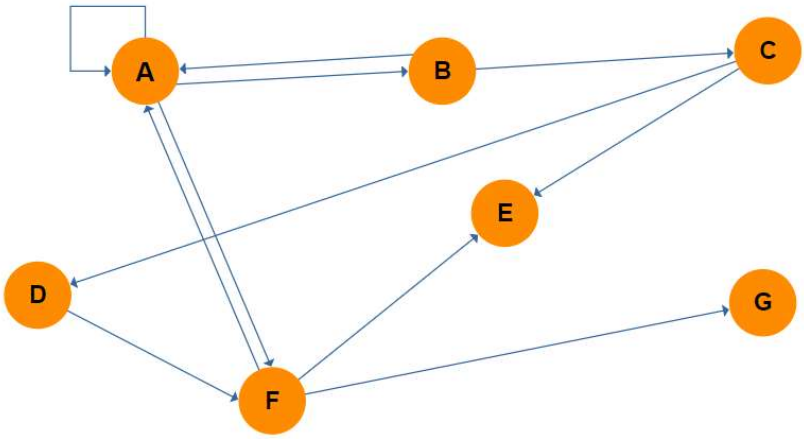

Abbildung 11: Ein exemplarischer Webgraph

6.1 Eingehende sowie ausgehende Links und ihre Verteilung

Man kann die Links unterscheiden in eingehende Links (engl. *in-links*) sowie ausgehende Links (engl. *out-links*). In dem obigen Beispiel besitzt der Knoten *F* 2 eingehende Links sowie 3 ausgehende Links. Alternativ spricht man auch von *Eingangsgrad* (engl. in-degree) bzw. *Ausgangsgrad* (engl. out-degree).

In größeren Graphen wie dem Webgraph kann eine Erhebung der Wahrscheinlichkeitsverteilung bzgl. der Anzahl der in-links interessant sein. Falls dabei jede Webseite als Ziel ihrer Links beliebige Seiten wählen würde, so würde sich die Wahrscheinlichkeitsverteilung der Anzahl der in-links einer *Binomialverteilung* annähern. Tatsächlich sind die Ziele von Verweisen nicht rein zufällig und damit ergibt sich eher eine Verteilung, die einem *Potenzgesetzt* folgt (engl. *Power Law)*: Die Wahrscheinlichkeit für eine Anzahl *i* von in-links entspricht 1/(i^α), wobei nach empirischen Studien für α ein Wert von etwa 2,1 angenommen werden darf. Der daraus entstehende Graph zeigt, dass die Wahrscheinlichkeiten für eine geringe Anzahl von in-links hoch ist und mit einer steigenden Anzahl von in-links stark abnimmt. Die Verteilungskurve kann damit statistisch als rechtsschief charakterisiert werden.

6.2 Verbundenheit des Webgraphs

Der Webgraph ist nicht stark verbunden miteinander. Das bedeutet, dass Paare bestehend aus jeweils zwei Knoten existieren, so dass man den einen Knoten weder direkt oder indirekt über den anderen Knoten erreicht. In dem obigen Beispiel wäre (*G, A*) so ein Paar, es existiert kein Pfad ausgehend von *G*, der per Link direkt oder indirekt über andere Knoten schließlich zu *A* führt.

In einer weiteren Studie (vgl. Broder et al. 2000) wurde die Struktur des Webgraphs charakterisiert bezüglich der Verbundenheit. Wie bereits das obige Beispiel zeigt, ist der Webgraph nicht stark miteinander verbunden. Nach dieser Studie, die in den folgenden Jahren im Wesentlichen bestätigt wurde, kann man sich den Webgraph in Form eines Querbinders vorstellen (engl. *bow-tie structure*). Nach diesem Schema besteht ein Anteil des Webgraphs von etwa 27% aus einer stark miteinander verbundenen Komponente, jeder Knoten kann darin durch jeden anderen direkt oder indirekt erreicht werden. Ein Anteil von etwa 21% gehört der *IN-Menge* an. Alle Knoten dieser Menge können die stark miteinander verbundenen Komponenten per Link erreichen, sie können jedoch nicht umgekehrt von der Menge der stark miteinander verbundenen Knoten erreicht werden. Ein Anteil von ebenfalls etwa 21% gehört der *OUT-Menge* an. Die Knoten dieser Menge sind erreichbar durch die stark miteinander verbundene Komponente, sie können jedoch nicht umgekehrt die stark miteinander verbundene Komponente erreichen. Es existieren zudem Kanäle, durch die die Knoten der IN-Menge die Knoten in der OUT-Menge direkt erreichen können. Zusätzlich existieren noch *Ranken*, die sich an die IN-Menge bzw. OUT-Menge anschließen und ausschließlich in die jeweils entgegengesetzte Richtung verweisen. In der IN-Menge kann man ihnen folgen und endet schließlich in einer Sackgasse. Analog dazu kann man ihnen folgen, um in die OUT-Menge zu gelangen.

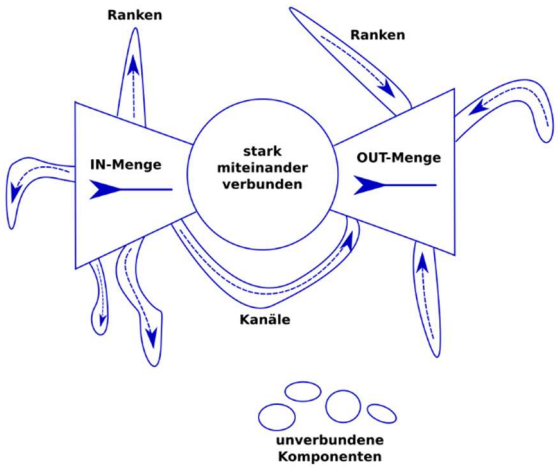

Abbildung 12: Bow-Tie Shape of the Web

Die globale Struktur des Web-Graphen kann man jedoch nicht direkt auf nationale Web übertragen, wie eine Studie (vgl. Donato et al. 2008: 3,4) gezeigt hat, nach diesen bestehen die nationalen Webs aus Italien, Indochina sowie dem UK fast ausschließlich aus einer Menge bestehend aus stark miteinander verbundenen Knoten sowie der OUT-Menge.

6.3 Dynamik sowie (potentielle) Unendlichkeit des Webgraphs

Darüber hinaus unterliegt der Web-Graph einer ständigen evolutionären Entwicklung, daher kann die Untersuchung sich nur auf eine Stichprobe beziehen, welche innerhalb eines gewissen Zeitraums gewonnen wurde.

Zudem ist zu beachten, dass das moderne Web durch dynamische Technologien eine potenziell unendliche Größe erreicht hat, indem etwa theoretische lediglich durch den Speicher begrenzt viele Instanzen von Webseiten autogeneriert werden können. Ebenfalls reagieren viele Webserver mit einem sogenannten *soft 404 error*, indem sie auf einen Request bzgl. einer nicht existenten URL mit einer gültigen HTML-Seite reagieren. Da potenziell theoretisch unendlich viele Strings generiert werden können würde jeder davon eine gültige Webseite generieren können (vgl. Manning et al. 2008: 433).

Eine weitere wichtige Eigenschaft des statischen Web-Graphs besteht darin, dass dort etwa 40% der Knoten bzw. Webseiten Duplikate darstellen. Dazu kommt ein Anteil von Webseiten mit beinahe-identischen Duplikaten, die etwa mittels Shingling als solche identifiziert werden können. Je nachdem wie der Web-Graph durchlaufen wird, kann zudem dynamisch eine sehr hohe Anzahl an Duplikaten entstehen, etwa durch automatische Generation von Seiten mit identischen Inhalten (vgl. Manning et al. 2008: 437).

7 Arten der Erhebung einer Stichprobe eines Graphens

Im Folgenden soll eruiert werden, wie man sinnvolle Stichproben eines Graphens gewinnen kann. Dabei geht es hier um Untersuchungen, die von den Eigenschaften innerhalb der Knoten abstrahieren und sich primär auf mathematische Eigenschaften des Graphens an sich fokussieren.

7.1 Ziele der Stichprobenerhebung

Leskovec und Faloutsos (vgl. Leskovec/Faloutsos 2006: 1) unterscheiden zwei verschiedene Ziele:

1. Die Stichprobe soll dem Ursprungsgraphen entsprechen mit zu diesem ähnlichen bzw. ggf. herunterskalierten Eigenschaften (*scale-down goal*)
2. Die Stichprobe soll dem Ursprungsgraphen ähnlich sein zu dem Zeitpunkt in der Vergangenheit, in dem der Ursprungsgraph dieselbe Größe wie die Stichprobe hatte (*back-in-time goal*)

7.2 Relevante Eigenschaften von Graphen

Es existieren verschiedene Algorithmen um interessante Messwerte über die Graphen erheben zu können, etwa den kürzesten Pfad zwischen zwei Knoten, man kann die bereits erwähnte Anzahl der eingehenden bzw. ausgehenden Links messen und so Knoten mit besonders vielen eingehenden bzw. ausgehenden Links eine hohe *Zentralität* innerhalb des Graphen bestätigen, andere Knoten können eine hohe *Betweenness* aufweisen, sie liegen also auf dem Weg von sehr vielen kürzesten Wegen zwischen Knoten und stellen damit ebenfalls selbst relevante Knoten dar. Ebenfalls kann man die Anzahl der Kanten auf dem kürzesten Weg zwischen den am weitesten entfernten Knoten messen und so ein Maß für den *Durchmesser* des Graphen bestimmen. Unter anderem aus der Anzahl der eingehenden Links wird auch der *PageRank* bestimmt, nach dem Knoten mit besonders vielen eingehenden Links, welche idealerweise wieder von Knoten mit einem ebenfalls hohen *PageRank* stammen, eine hohe Relevanz innerhalb eines Netzwerkes zugewiesen werden. Hier besitzen gerichtete Verbindungen eine Gewichtung und ein Verweis von einem Knoten mit hohem *PageRank* besitzt eine entsprechend höhere Gewichtung als Verweise von Konten niedrigerer *PageRanks*.

Leskovec und Faloutsos (vgl. Leskovec/Faloutsos 2006) folgern, dass man sich daher auf die Entnahme einer Stichprobe fokussieren sollte, da eine Erhebung des gesamten Graphens unwirtschaftlich bzw. unmöglich würde.

Daraus ergeben sich für sie die folgenden Fragestellungen:

1. Welche Methode der Stichprobenentnahme sollte gewählt werden?
2. Wie klein kann die Stichprobe sein?
3. Wie können die gewonnenen Messwerte hochskaliert werden, beispielsweise um den Durchmesser des tatsächlichen Graphen abzuschätzen?

Nach Leskovec und Faloutsos (vgl. Leskovec/Faloutsos 2006) kann man die Algorithmen zur Stichprobenentnahme konzeptuell in 3 Gruppen einordnen:

1. Methoden basierend auf einer zufälligen Auswahl von Knoten
2. Methoden basierend auf einer zufälligen Auswahl von Kanten
3. Explorationstechniken die Irrfahrten (engl. random walks) oder eine Virus Ausbreitung simulieren.

Als Fazit ihrer Untersuchungen zeigen sie, dass für das *scale-down goal* Irrfahrten die besten Ergebnisse liefern, da sie eine Präferenz hin zu Knoten mit einer hohen *Gradzahl* aufweisen und Stichproben entnehmen, die zusammenhängende Graphen darstellen wohingegen für das *back-in-time goal* die Methoden *Forrest Fire* sowie eine Stichprobenentnahme basierend auf dem *PageRank* von Knoten am geeignetsten sind. Nach den *Forest Fire* Methoden, die der Ausbreitung von größeren Waldbränden nachempfunden sind, beginnt man sich von mehreren Brantnestern=Ausgangspunkten aus durch den Graphen zu bewegen, wobei ein Baum=Knoten mit einer gewissen Wahrscheinlichkeit brennt, sobald ein Nachbar, also ein anderer Knoten mit einer direkten Verbindung zu dem ursprünglichen Knoten auch brennt. Dies setzt sich rekursiv fort. Mit diesen Methoden können statische sowie dynamische Muster der Graphen präzise durch die Stichprobe abgebildet werden, wobei die Stichproben auf bis zu 15% der Größe des ursprünglichen Graphen heruntergehen können. Andere Suchstrategien wie etwa die systematische Tiefen- und Breitensuche konnten keine ähnlich guten Ergebnisse erzielen.

Man kann man die Explorationstechniken von Leskovec und Faloutsos wie Irrfahrten und Virus Simulation nochmals weiter untergliedern in Verfahren, in denen identische Knoten wiederbesucht werden können wohingegen Explorationstechniken wie die Tiefen- oder Breitensuche dies nicht gestatten.

8 Metadaten von XML- und HTML-Dokumenten

Das `meta`-Tag gehört zu den Kopfdaten von HTML-Dokumenten. Es bietet die Möglichkeit, Metadaten über das Dokument vorzuhalten, welche in keinem anderen Tag mit semantischem Inhalt wie beispielsweise `<title>` hinterlegt sind. Die Anzahl der zu nutzenden Meta-Elemente ist beliebig, der typische Aufbau dieser Elemente lautet:

```
<meta property= "value">
```

Typische Eigenschaften sind der Zeichensatz (engl. `charset`), eine Beschreibung (engl. `description`), Schlüsselwörter (engl. `keywords`) sowie Informationen zur Darstellung auf einem Display (engl. `viewport`). Diese Einstellungen sind besonders für mobile Endgeräte relevant, so dass mit geeigneten Parametern eine Darstellung auf dem Viewport gewährleistet ist, ebenfalls aber skaliert werden kann. Ebenfalls finden sich häufig Informationen für den Webcrawler wieder (engl. `robots`).

Ursprünglich sollten die Metadaten den Suchmaschinen bei der Klassifikation der Ressourcen helfen, häufig wurde diese Eigenschaft jedoch missbraucht, so dass in Folge die Bedeutung der Metadaten nur noch ein Faktor unter vielen bei der Klassifikation sowie Einordnung der Relevanz einer Ressource darstellt.

Ein weiteres grundsätzliches Problem besteht in der fehlenden Standardisierung von Metadaten über die wenigen relevanten obigen Eigenschaften hinaus. Durch die beliebige Ausgestaltung von Metadaten wird die automatische Verarbeitung dieser seitens der Algorithmen schwierig. Stattdessen existieren Profile bereits vordefinierter und damit standardisierter Metadaten. Beispiele für solche Profile beziehungsweise Ontologien sind *Schema.org*, sowie die *Dublin Core Metadata Initiative*. *The Open Graph protocol* stellt ebenfalls ein Schema bereit, welches auch insbesondere von Facebook und Twitter genutzt wird.

Alle diese Profile finden sich auch in den gewonnenen Metadaten wieder.

9 Realisierung der eigenen Ideen & Konzepte – Versuchsaufbau

In diesem Abschnitt wird es um die Umsetzung der Ideen gehen. Zunächst wird kurz der gesamte Aufbau des Systems zur Gewinnung sowie Auswertung der Stichprobe dargestellt. Es schließen sich genauere Betrachtungen der einzelnen Systemelemente an. Es wird ebenfalls auf dabei auftretende Probleme sowie deren aktuellen Lösungszustand eingegangen:

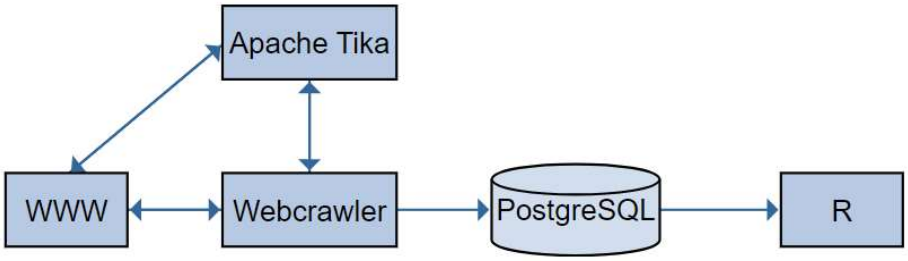

Abbildung 13: Systemaufbau

9.1 Entwicklung der Webcrawler

Die selbst erstellen Webcrawler basieren ursprünglich auf dem *Site Graph Skript* von Kiran Tomlison, geschrieben in der Programmiersprache Python (vgl. Cornell University/Tomlinson o. D.). Die Aufgabe seines Skriptes besteht darin, die Zusammenhänge auf einer Website zu visualisieren, daher werden nur ausgehende Links von der darzustellenden Website untersucht, um anschließend einen graphischen Output des so entstandenen Graphs zu erstellen. Es werden sowohl XML- und HTML-Dokumente als auch andere Ressourcen wie beispielsweise PDFs gecrawlt, jedoch werden nur XML- und HTML-Dokumente genutzt, um weitere Links zu extrahieren und sich entlang ihnen zu bewegen. Die gewählte Suchstrategie ist eine *Tiefensuche*, eine hier sinnvolle Strategie, da immer lediglich von einer Website ausgegangen wird und die verlinkten Ressourcen ihrerseits Endpunkte darstellen, die zwar selbst noch untersucht werden, denen jedoch nicht weiter nachgegangen wird. Das Fetching wird durch die Python Bibliothek `requests` durchgeführt, eine einfach gestaltete Bibliothek für HTTP Requests. Das Parsen der XML- sowie HTML-Dokumente erfolgt durch die *Beautiful Soup*, welche eine mächtige Bibliothek für das Parsen von XML- sowie HTML-Dokumenten darstellt. Kiran Tomlinson beschreibt selbst einige wesentliche Bestandteile für ein funktionierendes Crawling (vgl. Tomlinson 2020). Einige der Herausforderungen waren das Auflösen von Links, insbesondere von relativen Verweisen, die sich auf eine Basis URL beziehen, die nicht der Basis URL entspricht, die unter dem `<base>` Tag ihres XML- sowie HTML-Dokuments angegeben wird. Weitere Probleme lagen in der Standardisierung von URLs, der Umgang mit Fehlern beim Fetching, eine Minimierung der herunterzuladenden Daten sowie der Umgang von in der URL eingebetteten mit Queries sowie Fragmentbezeichnern.

9.1.1 Entwicklung des Webcrawlers mit Breitensuche mit einem Startpunkt

Der vollständige Sourcecode des Webcrawlers befindet sich im Anhang. Zunächst wurde der Webcrawler so verändert, dass als Suchstrategie eine Breitensuche genutzt wird. Realisiert durch eine `deque`, eine *Double Ended Queue* aus dem Modul `collections`. Anschließend wurde die Suchstrategie so angepasst, dass die Tiefe der Suche auf eine festgelegte Anzahl von Ebenen begrenzt wurde. Darauffolgend wurde *Apache Tika* mit eingebunden, um von den gecrawlten Ressourcen die Metadaten abrufen und in eine *PostgreSQL* Datenbank abspeichern zu können. Da die Absicht darin besteht, von einer Ressource sämtliche Metadaten

zu ermitteln, wurde ein Parser von Apache Tika entsprechend konfiguriert, er liefert als Ergebnis ein Objekt vom Typ Python *Dictionary* zurück, welches einem JSON Objekt gleicht. Dieses wird, nach einer Konvertierung in ein für die Datenbank verarbeitbares JSON Objekt, zusammen mit der URL der Ressource in die PostgreSQL Datenbank gespeichert. Da es beliebige Ausprägungen von Metadaten je Ressource geben kann ist es ungeeignet, ein festes Schema für die einzelnen Ausprägungen der Metadaten in der Datenbank als jeweils eigenständige Tabellenspalten anzulegen, daher besitzt die Tabellenspalte, in der die Metadaten gespeichert werden, den Datentyp `jsonb`. Nähere dazu im kommenden Abschnitt über das Schema der Datenbank. In dem ursprünglichen Crawler von Kiran Tomlison wurde jeweils zunächst mittels eines HTTP Head Requests geprüft, ob eine verlinkte Ressource abrufbar ist und anschließend nur die Ressourcen mittels HTTP Get Requests abgerufen, deren Typen man auch untersuchten möchte, basierend auf dem bereits im HTTP Head Request übermittelten MIME Type. Diese Strategie wurde gewählt, um den entstehenden Netzwerktraffic zu reduzieren. In dem umgestalteten Webcrawler wurde dagegen die Strategie gewählt, direkt mittels HTTP Get Requests sämtliche Informationen abzurufen. Da weiterhin Beautiful Soup genutzt wird, werden im Folgenden durch die Crawler auch nur noch Ressourcen in Form von XML- sowie HTML-Dokumenten untersucht. Da die HTTP Head Requests ebenfalls mit der Netzwerklatenz belastet sind, wurde sich in der Strategie des eigenen Crawlers für einen tendenziell höheren Netzwerktraffic entscheiden, da von jeder Ressource die gesamten initialen Informationen übertragen wurden und darauffolgenden ggf. verworfen werden, dafür aber mit einer geringeren Anzahl von Requests durch den Wegfall der HTTP Head Requests entschieden. Bezüglich der nachfolgenden Betrachtungen ist noch anzumerken, dass dieser Webcrawler einen garantiert mittels Hyperlinks verbundenen Teilausschnitt des WWWs zurückliefert. Dies ist bei der Suchstrategie mit einer Breitensuche basierend auf mehreren Startpunkten nicht mehr zwingend gewährleistet.

9.1.2 Entwicklung des Webcrawlers mit Breitensuche mit mehreren Startpunkten

Nach einigen Testläufen mit dem Webcrawler ausgehend von nur einem Startpunkt zeigte sich der Flaschenhals in Form von HTTP Requests deutlich. In einem ersten Zwischenschritt wurde der Crawler dahingehend angepasst, dass der Crawlvorgang in Form eines parametrisierten Funktionsaufrufs umgestaltete wurde, wobei die `crawl()` Funktion jeweils eine Menge (=Set) von besuchten XML- sowie HTML-Dokumenten sowie die Kanten zwischen diesen Ressourcen in Form von Hyperlinks zurückliefert. Diese zurückgegebenen Daten werden anschließend von dem aufrufenden *Main Thread* so vereinigt, dass Doppelungen dabei entfernt werden. Anschließend werden die so bereinigten Daten von Apache Tika geparst und in die Datenbank gespeichert.

In dem nächsten Entwicklungsschritt wurde der Webcrawler nebenläufig gestaltet. Dazu wurde das objektorientierte Modul `threading` genutzt, welches keine echte Parallelisierung auf mehreren Prozessorkernen gestattet, sondern mit *Threads* arbeitet, die in einem gemeinsamen Speicherraum liegen und für das OS einfach zu verwalten sowie weniger ressourcenaufwendig als Prozesse sind. Sie werden daher auch *Leichtgewichtprozesse* genannt. Da sie in Python gegenwärtig nur auf einem Prozessorkern arbeiten, sind sie für die Durchführung komplexer Berechnungen und ähnliche Aufgaben ungeeignet, wohingegen man sie nach allgemeiner Empfehlung sinnvoll für Aufgaben mit hohen Latenzen, wie sie beispielsweise bei Netzwerkverbindungen entstehen, einsetzen kann. Alternativ existiert noch das Modul `multiprocessing`, mit welchem echte verschiedene Prozesse angelegt und parallel auf mehreren Prozessorkernen ausgeführt werden können. Hier existiert jedoch kein gemeinsam genutzter Speicherraum mehr, ein Austausch zwischen den Prozessen wird schwieriger.

In dem Crawler wird die Nebenläufigkeit mittels Threads realisiert, damit wird die bereits im Kapitel über das Crawlerdesign erwähnte Strategie verfolgt, nach der mehrere Threads existieren, wobei innerhalb von ihnen jeweils mit seriellem I/O gearbeitet wird.

Die mit dem Webcrawler gewonnene Erfahrung zeigt jedoch, dass ebenfalls eine sehr hohe Prozessorauslastung stattfindet. Ursächlich dafür könnte sein, dass mit großen nicht-persistenten Datenstrukturen gearbeitet wird, die innerhalb des Arbeitsspeichers liegen, da zunächst erst nach einem Crawlvorgang die gewonnenen Daten in die Datenbank persistent gespeichert werden. Dies liegt darin begründet, dass die Threads nach aktuellem Entwicklungsstand nicht untereinander kommunizieren und so ggf. identische Ressourcen mehrfach besuchen. Erst nachdem alle Threads die vorgegebene Anzahl von Ebenen gecrawlt haben und die Ergebnisse ihrer Suche in Form von threadlokalen Datenstrukturen zurückgeben, fand zunächst eine mengenmäßige Vereinigung der Ergebnisse statt, um Doppelungen zu entfernen und anschließend die so bereinigten Daten in der Datenbank persistent zu speichern. Eine weitere Ursache für die hohe Prozessorlast könnte auch in der Anzahl der Threads und dem damit verbundenen Verwaltungsaufwand für das OS liegen, auch sind die genutzten Python Bibliotheken zwar bequem nutzbar und damit für ein rasches Prototyping geeignet, jedoch nicht die effizientesten Implementierungen für alle Problemstellungen.

9.1.3 Entwicklung des Webcrawlers mit Random Walk Verhalten

Gemäß den gewonnenen Erkenntnissen von Leskovec und Faloutsos, dargestellt im Kapitel 10.2, stellt die Irrfahrt bzw. der Random Walk eine geeignete Methodik dar, um das *scale-down goal* zu erreichen. Begründet wird dies mit der höheren Wahrscheinlichkeit, dabei auf relevante Webseiten mit einem hohen In-Degree zu stoßen. Auch wenn die Aufgabe in dem Clustern der gewonnenen Metadaten besteht, so kann man ggf. annehmen, dass ein Zusammenhang zwischen dem In-Degree und der Repräsentativität der Ressource besteht. Damit würde eine mittels der Random-Walk-Strategie gewonnene Stichprobe auch eine hohe inhaltliche Repräsentativität aufweisen. Zu dem jetzigen Zeitpunkt stellt diese Annahme jedoch zunächst eine reine Hypothese dar. Daher wurde neben den beiden bisherigen Webcrawlern noch ein dritter Webcrawler entwickelt, welcher basierend auf der Random-Walk-Strategie arbeitet. Es existieren verschiedene Varianten dieser Strategie. Gewählt wurde hier die Variante, bei der bereits besuchte XML- und HTML-Dokumenten wieder besucht werden dürfen, dieses Verhalten entspricht der wahrscheinlichkeitstheoretischen Strategie einer *Ziehung mit Zurücklegen*. Der Hintergrund für die Wahl dieser Strategie besteht darin, dass der Random Walk so mit einer höheren Wahrscheinlichkeit nicht in einer Senke endet.

9.1.4 Bei der Crawlerentwicklung aufgetretene Probleme

Unabhängig von den einzelnen Suchstrategien haben sich bei der Entwicklung sowie dem Testen der Crawler einige relevante Probleme ergeben. Hierbei war gestaltete es sich in der Praxis insbesondere schwierig, dass diese Probleme häufig erstmalig nach bereits lang andauernden Crawlvorgängen auftraten.

9.1.4.1 JSON mit Unicode Codepunkt \u0000

Ein mit PostgreSQL aufgetretenes Problem besteht in dem Vorkommnis des Unicode Codepunktes \u0000. Falls dieser in dem von Apache Tika zurückgelieferten Python Dictionary enthalten ist, wurde die Exception `ERROR: unsupported Unicode escape sequence Detail: \u0000 cannot be converted to text` bei dem Versuch geworfen, das aus diesem Python Dictionary mittels des PostgreSQL Treibers `psycopg2` über seine Wrapper Klasse in ein JSON Objekt umgewandeltes Objekt in PostgreSQL zu speichern. Dieser Unicode Codepunkt ist in einem String bzw. in einem JSON Objekt in PostgreSQL nicht gestattet. Würde man es gestatten, diesen Codepunkt in einem String bzw. JSON Objekt zu nutzen, so entstünden Probleme mit auf der Programmiersprache C basierten APIs, in welchen \0 als terminierendes Element von Strings interpretiert wird. Auch die im Internet vorgeschlagene Lösung, mittels regulärer Mustersuche den Codepunkt einfach aus dem String zu entfernen, bevor man ihn in PostgreSQL abspeichert,

hat das Problem nicht gelöst. Nach aktuellem Stand wird daher lediglich der String *Parserfehler* für die entsprechenden Ressourcen in der Datenbank vermerkt.

9.1.4.2 URLs mit finalem `'/'`

Dieses Problem liegt in dem Verhalten von Apache Tika begründet. Falls Apache Tika URLs mit einem finalem `'/'` parsen soll, wird die Fehlermeldung `IsADirectoryError: [Errno 21] Is a directory: '/tmp/'` ausgegeben. Das Problem rührt daher, dass eine URL mit finalem `'/'` als Verzeichnis angesehen werden kann. In dem WWW antworten Webserver unterschiedlich auf solche Requests. Einige liefern dafür auch ein Verzeichnis zurück, viele Webserver liefern aber auch stattdessen eine Datei zurück, welche beispielsweise bei dem Apache Webserver mittels der Option `DirectoryIndex` angegeben werden kann. In meinen Webcrawlern werden finale `'/'` gegenwärtig abgeschnitten.

9.1.4.3 Exception socket.timeout: The read operation timed out

Die Webcrawler nutzen die `requests` Bibliothek von Python. Um nicht zu lang auf die Antwort eines Webservers warten zu müssen wird der HTTP GET Requestaufruf mit einem Timeout von 10 Sekunden parametrisiert. Intern liegt die `requests` Bibliothek eine Abstraktionsebene über der `urllib3` Bibliothek und baut intern auf sie auf. Das auftretende Problem besteht darin, dass wenn diese Exception geworfen wird, sie nicht über den Exception-Handling-Mechanismus der `requests` Bibliothek selbst abgefangen werden kann. Die Ursache besteht darin, dass die Exception eigentlich durch die intern von `requests` genutzte Bibliothek `urllib3` geworfen wird, die Exception jedoch nicht ordnungsgemäß als eine `requests` Exception verpackt wird, sondern explizit als eine Exception der `urllib3` Bibliothek mittels `(requests.exceptions.RequestException, socket.timeout)` abefangen werden muss (vgl. Socket.timeout is thrown instead of requests.exceptions.Timeout Issue #1236 psf/requests 2013). Dieses Verhalten verletzt das Schichtenprinzip der Abstraktionsebenen.

9.1.4.4 Herausfiltern der korrekten Top-Level-Domain (=TLD)

Die korrekte TLD ohne ggf. vorhandene Subdomains herauszufiltern kann ein Problem darstellen, da keine intrinsische Möglichkeit existiert, eine Subdomain zu erkennen. Das Problem hängt damit zusammen, dass verschiedene nationale URL Registrare unterschiedliche Arten von Domains verkaufen. So stellt beispielsweise *zap.co.it* eine Subdomain dar, da italienische URL Registrare Domains wie *co.it* verkaufen, wohingegen in dem United Kingdom nationale Registrare nur Domains wie beispielsweise *zap.co.uk* verkaufen, und damit *zap.co.uk* keine Subdomain darstellt. Die von mir gewählte Lösung besteht darin, dass Python Modul `tld` zu nutzen, welches sich intern auf eine öffentliche Suffix Liste abstützt, welche von der Mozilla Foundation gepflegt wird (vgl. How to extract top-level domain name (TLD) from URL 2009).

9.1.4.5 Crawlerfalle in Form von automatisch generierten Session IDs

Bei den ersten Crawlvorgängen ist die Situation entstanden, dass der Crawler auf ausgelieferten HTML-Dokumenten gefangen geblieben ist. Diese Dokumente wurden dynamisch generiert und zeichnen sich dadurch aus, dass jeweils neue Sessions mit eigener ID generiert werden, in dieser Schleife bleibt der Crawler dann gefangen (vgl. Kapitel 7.4.5). Eine Lösung besteht darin, die URLs erst zu normieren, indem vermittels der Funktion `filtere_url()` unter Anderem die Elemente der URL abgeschnitten werden, welche auf die Session ID verweisen. Der Vergleich dieser so bereinigten URL mit der Menge der bereits gesehenen URLs verhindert dann, dass eine final identische Ressource, die sich nur durch die Session ID unterscheidet mehrmals besucht wird.

9.1.4.6 Töten des Crawlerprozesses durch das OS

Da die gewonnenen Daten über die entdeckten Webseiten sowie der Verlinkungen zwischen ihnen zunächst ausschließlich in Datenstrukturen innerhalb des Arbeitsspeichers vorgehalten wurden, ergab sich die Situation, dass der Crawlprozess, wenn sich die Auslastung des Arbeitsspeichers gegen 100% bewegte, durch das OS getötet wurde. Damit waren auch die bisherigen Daten verloren. Diese Strategie wurde ursprünglich verfolgt, da wie bereits dargestellt, über die mengenmäßige Vereinigung der Daten innerhalb des parallelen Crawlens erst nach dem Vollenden aller Crawlvorgänge Doppelungen entfernt wurden. Die finale angewendete Lösung besteht nun darin, für sämtliche Varianten des Crawlers die Daten in einer temporären Datenstruktur innerhalb des Arbeitsspeichers vorzuhalten, um einen schnellen Abgleich darüber zu ermöglichen, welche XML- sowie HTML-Dokumente bereits entdeckt wurden, jeweils bezogen nur auf die Informationen des eigenen Threads. Zusätzlich werden jeweils unabhängig voneinander die entdeckten XML- sowie HTML-Dokumente sowie die Kanten zwischen ihnen auch in die Datenbank gespeichert. Im Falle der Breitensuche mit nur einem Einstiegspunkt können hierbei auch keine Konflikte auftreten, da mittels der Datenstrukturen bereit sichergestellt wird, dass sämtliche zu speichernde Informationen `UNIQUE` sind. Im Falle des parallelen Crawlens mittels Threads ist dies jedoch bisher nicht gegeben, da nicht über die verschiedenen Threads hinweg kommuniziert wird bzw. eine einheitliche Datenstruktur die zu crawlenden Daten koordiniert an sie verteilt. Daher werden hier gegenwärtig auch weiterhin identische Ressourcen mehrfach besucht und ggf. zusätzliche Netzwerklast produziert. Jedoch kann über den Einfügevorgang auf der Datenbank sichergestellt werden, dass identische Informationen nicht mehrfach vorgehalten werden. PostgreSQL bietet die Möglichkeit eines nur bedingten Einfügevorgangs. Dieser stellt sich im Programmcode folgendermaßen dar:

```
("INSERT INTO webseite (url, metadaten) VALUES (%s, %s) ON CONFLICT (url) DO NOTHING", [ in_url, Json(res) ])
```

Mittels dieser Qualifizierung werden nur Datensätze eingefügt, welche keinen Konflikt mit bereits bestehenden Datensätzen bezogen auf das Attribut `url` besitzen. Dieses Attribut ist in der DB dabei mit einem `UNIQUE` Constraint belegt. Analog dazu werden auch identische Kanten zwischen Webseiten garantiert nur einmal in der Datenbank gespeichert.

9.1.5 Zukünftiges Optimierungspotential der Webcrawler

Die Webcrawler wurden von Beginn an mit dem primären Ziel entwickelt, als Mittel zum Zweck der Gewinnung von Stichproben mit verschiedenen Suchstrategien zu dienen. Daher ist das Programmdesign an einigen Stellen verbesserungswürdig. So halten nach wie vor aktuell alle Webcrawler die gewonnenen Daten in Datenstrukturen innerhalb des Arbeitsspeichers vor. Auch wenn die während des Crawlvorgangs gewonnenen Daten bereits persistent gespeichert werden, kann der Crawlvorgang nach wie vor wie bereits dargestellt durch das OS getötet werden. Eine alternative und perspektivisch bessere Implementierung kann darin bestehen, auch die Daten aus der DB abzurufen, um zu entscheiden, ob eine Ressource bereits entdeckt wurde. Dabei müssten auch die Kosten für eine Lese- und Speicheroperationen auf der Datenbank berücksichtigt werden. Eine noch bessere Lösung wurde bereits im Kapitel über das Design des Webcrawlers Mercator dargestellt, indem man häufig abgefragte Daten in einem Cache vorhält und die anderen Datenbestände auf der Festplatte respektive in der Datenbank nachfragt.

Im Falle des parallelen Crawlens besteht ein weiteres Problem im Umgang mit dem mehrfachen Crawlen bereits entdeckter Ressourcen. Nach aktueller Logik können identische Ressourcen unabhängig voneinander mehrfach gecrawlt werden, theoretisch kann jeder Thread dieselbe Ressource crawlen. Stattdessen sollte eine Strategie gemeinsamer Datenstrukturen genutzt werden, wie sie auch im Kapitel über das Design des

Webcrawlers Mercator vorgeschlagen wird, nach der die zu crawlenden Ressourcen von einer zentralen Instanz an die einzelnen Crawlerthreads verteilt werden und darüber dieses Problem ausgeschlossen wird.

Weiterhin handelt es sich bisher um keine „freundlichen“ Webcrawler, da das *Robots Exclusion Protocol* nicht beachtet wird. Nach diesem Protokoll muss zunächst innerhalb des Wurzelverzeichnisses aus der zu crawlenden Domain eine ggf. vorhandene Datei `robots.txt` ausgelesen werden. In ihr ist vermerkt, welche Teile der Domain gecrawlt werden dürfen. Zusätzlich findet sich diese Information häufig auch innerhalb der Metadaten von XML- sowie HTML-Dokumenten.

9.2 Apache Tika

Apache Tika stellt ein Framework für die Inhaltsanalyse sowie Inhaltsdetektion dar. Es ist sowohl in der Lage, Inhalte als auch Metadaten auszulesen. Die besondere Stärke liegt in der Vielzahl der Datentypen (über 1400) begründet, von denen Tika in der Lage ist, Metadaten auszulesen bzw. zu erschließen.

9.3 Schema der Datenbank

Nachfolgend wird der einfache Aufbau der PostgreSQL Datenbank dargestellt in Form des Entity-Relationship-Modells sowie des zugehörigen Relationenmodells inklusive der genutzten Datentypen:

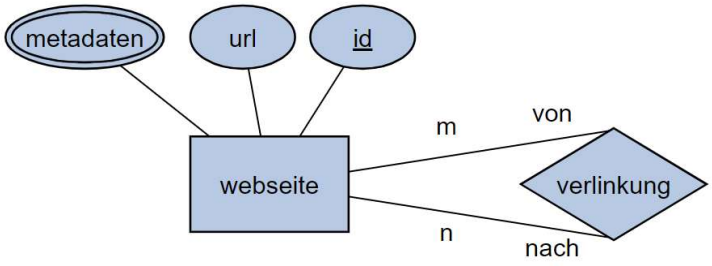

Abbildung 14: Aufbau DB - ERM

Der Datenbank kommt in diesem System eine untergeordnete Rolle zu. Bereits durch die eingesetzten Datenstrukturen wird gewährleistet, dass keine identischen URLs mehrfach auftreten können sowie zwischen je zwei Webseiten immer nur genau eine gerichtete Beziehung existieren kann. Damit entfällt eine der typischen Aufgaben einer Datenbank, die Sicherstellung der Integrität. Die eigentliche Komplexität liegt hinter der Verlinkungsstruktur sowie den Metadaten verborgen. Eine komplexere Verlinkungsstruktur ist mit den Mitteln von SQL nicht leicht abbildbar, man benötigt beispielsweise fortgeschrittene Methoden wie rekursive Queries, um Ketten aus Verlinkungen darstellen zu können bzw. um festzustellen, ob Pfade zwischen je zwei Knoten existieren und beispielsweise, welche davon die Kürzesten darstellen. Graphenbasierte Datenbankmanagementsysteme wie Neo4j ermöglichen einfach solche Auswertungen. Da die spätere Auswertung sich aber zunächst auf die Metadaten fokussieren wird sowie da für die Programmiersprache *R* auch Bibliotheken für graphenbezogene Fragestellungen existiert, stellt die weniger komfortable Auswertungsfähigkeit einer objektrelationalen Datenbank wie PostgreSQL hier zunächst keinen Nachteil dar.

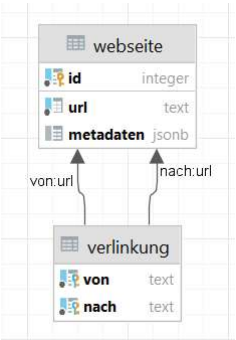

Abbildung 15: Aufbau DB - RM

Gemäß dem Relationenmodell der Datenbank wird zudem für jede Webseite ein Surrogatschlüssel generiert. Das mehrwertige Attribut `metadaten` besitzt den Datentyp `jsonb`. In diesem Datentyp werden die Daten in binärer Form gespeichert, anstelle der Speicherung in Form eines UTF-8 Strings, wie dies im Datentyp `json` geschieht. Wesentliche Vorteile liegen in der effizienteren Verarbeitungsfähigkeit, insbesondere können Spalten dieses Typs indexiert werden. Dies kann relevant werden, falls man die Stichprobengröße stark erweitern möchte und beispielsweise eine Vorauswahl der später zu verarbeitenden Datensätze bereits vor dem Import in R oder einer anderen Umgebung über die Datenbank vornehmen möchte. Möchte man eine Vorauswahl basierend auf Eigenschaften innerhalb einer Spalte des Datentyps `jsonb` vornehmen, so können sich dank der Indexierung Geschwindigkeitsvorteile mit einem Faktor bis zu 1000 gegenüber den nicht indexierbaren Spalten des Datentyps `json` ergeben.

9.4 R

Zu der weiteren Aufbereitung und Harmonisierung wurde sich für die Programmiersprache R und die freie IDE *rstudio* entschieden. Beweggrund für die Entscheidung ist, dass es zahlreiche leistungsstarke Bibliotheken rund um das Thema des Clustering gibt. Zudem existieren einfache Lösungen, die jeweils heterogen aufgebauten JSON Objekte tabellarisch in Form von Spalten zu glätten.

10 Realisierung der eigenen Ideen & Konzepte – Versuchsdurchführung

In diesem Abschnitt soll den eingangs gestellten Forschungsfragen nachgegangen werden. Dabei werden sowohl die Ergebnisse der Untersuchung dargestellt als auch auszugsweise die dafür relevanten Befehle. Zudem wird jeweils ein Bezug auf die Eignung für eine Gruppierung bzw. ein Clustering hergestellt. In den Anhängen befinden sich die jeweils vollständigen Listings, in denen die Schritte zusammenhängend und auch inklusive der hier unerwähnten, weil weniger relevanten, Zwischenschritte, dargelegt werden.

10.1 Crawlvorgang

Das Problem besteht bei den Crawlvorgängen, die nur auf einem Startpunkt basieren in der Wahl eines geeigneten Einstiegspunkts. Da es schwierig erscheint ein HTML-Dokument auszuwählen, welches insgesamt repräsentativ ist, wurde entschieden, den Crawlvorgang mit Breitensuche von der Webseite `https://www.spiegel.de` zu beginnen. Den Crawlvorgang basierend auf Random Walk wurde ebenfalls gestartet von der Webseite `https://www.spielge.de`. Für den parallelen Crawlvorgang mit der Breitensuche wurden 55 verschiedene Startpunkte ausgewählt aus der Liste der 100 am häufigsten in Deutschlang im Jahr 2022 aufgerufenen Webseiten (vgl. Sereda 2021). Dabei wurde unter diesen Webseiten gewählt, welche auch eine geeignete TLD aufweisen und zudem verschiedenen Bereichen wie beispielsweise Nachrichten, Sport und Anzeigenseiten. Die Idee dahinter ist, wie im Kapitel 6 über Stichproben dargelegt, eine möglichst hohe Repräsentativität zu erzielen. Zukünftige könnte man versuchen, das Verhältnis der Anzahl der Startknoten aus den verschiedenen Bereichen dem Verhältnis innerhalb des gesamten abgegrenzten Untersuchungsbereichs anzugleichen, sofern solche Informationen vorliegen. Inwiefern eine solche initiale Konfiguration aber auch innerhalb des darauf aufbauenden Crawlvorgangs die Verhältnismäßigkeit wahrt und ob dies überhaupt wünschenswert ist oder einen Fehlschluss der voreingenommenen Statistik darstellt, müsste ebenfalls beachtet werden.

Die Gesamtanzahl der gecrawlten Webseiten unterscheidet sich gegenwärtig stark voneinander, da der Crawler mit paralleler Crawlstrategie deutlich mehr Webseiten besucht und da gegenwärtig auch noch alle Crawlvorgänge laufen, wurde entschieden, zunächst jeweils aus dem gegenwärtigen Fundus je Crawlvorgang zufällig 5000 Datensätze auszuwerten.

Dies geschieht innerhalb von R nach dem Verbindungsaufbau zu der Datenbank durch den folgenden SQL-Befehl:

```
webseiten <- dbGetQuery(conn, "SELECT * FROM webseite ORDER BY random();")
```

10.2 Ergebnisse des Crawlvorgangs mit Breitensuche und einem Ausgangspunkt

Aus den 5000 zufälligen Webseiten mit ihren Metadaten entsteht ein Datenrahmen (engl. *Data Frame*) mit insgesamt 626 Spalten/Attributen für ihre Metadaten. Diese hohe Anzahl an Attributen entsteht dadurch, dass im Rahmen des Importvorgangs für jedes von Apache Tika geparste Metadatum mit seiner Ausprägung eine Spalte angelegt wird. Falls ein Metadatum mit einer bestimmten Bezeichnung also beispielsweise nur einmal so unter allen Metadaten vorkommt, so erhält dieses Datum trotzdem eine eigene Spalte in dem Datenrahmen in R. Für alle anderen Zeilen, in denen es für dieses Attribut keine Ausprägung gibt, wird automatisch das Symbol `NA` gesetzt. Dadurch ergibt sich eine insgesamt sehr breite Tabelle, die jedoch überwiegend aus dem Symbol `NA` besteht.

Da Attribute, die nur von wenigen bzw. ausschließlich durch einzelne HTML-Dokumente genutzt werden, auch für eine mögliche spätere Gruppierung bzw. ein Clustering im Allgemeinen nicht relevant sind, wurden sie vor der weiteren Analyse aus dem Datenrahmen entfernt. Ein Beispiel für solche durch Apache Tika gefundene Attribute ist der *Beschleunigungsvektor* (engl. *acceleration vector*), welcher aus dem Format eines eingebetteten Bildes übernommen wurde.

Mittels des Befehls wurden aus dem ursprünglichen Datenrahmen nur diejenigen Attribute herausgefiltert, die eine relevante Anzahl von Belegungen vorweisen können:

```
webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = c(id, url.x,
title, author, robots, Robots, ROBOTS, keywords, Keywords, description,
`Content-Type`, `Content-Encoding`, `Content-Language`, `content-language`,
`Content-language`, `og:type`, generator, Generator))
```

Als Schwellwert für die weitere Betrachtung wurde festgelegt, dass ein Attribut bei mindestens 15% der Webseiten vorhanden und mit einer Ausprägung belegt ist. Diese niedrige Schwelle ergibt sich aus dem Umstand, dass nur sehr wenige Attribute häufig genutzt werden.

Ein weiteres Problem, bereits durch den obigen Befehl ersichtlich, besteht darin, dass die Benamung von einzelnen Metadaten, falls sie nicht aus einem festen Benamungsschema wie der *Dublin Core Metadata Initiative* entstammen (vgl. Kapitel 11), keiner Normung unterliegt. Daher unterscheidet sich ihre Bezeichnung in der Schreibweise. So fanden sich beispielsweise für die Metadatenbezeichnung `robots` auch noch die abweichenden Schreibweisen `Robots` sowie `ROBOTS` in dem gecrawlten Datenbestand wieder. Da die überwiegende Anzahl von Merkmalsausprägungen unter der Bezeichnung `robots` vorliegt, wurde diese auch beibehalten und die Ausprägungen, die sich unter den anderen Bezeichnungen wiederfinden in die Spalte kopiert. Es existieren auch seltene Fälle, in denen eine identische Ressource Informationen über ein Metadatum unter verschiedenen Schreibweisen anbietet, sich aber die jeweiligen Merkmalsausprägungen unterscheiden. Dabei konnte stichprobenartig keine sich widersprechende Metainformationen gefunden werden, jedoch verschieden spezifische Ausprägungen. In solchen Fällen werden die Metainformationen, die unter der überwiegend genutzten Schreibweise gefunden werden, beibehalten. Überwiegend findet jedoch jeweils nur die Angabe der Information unter einer Schreibweise statt.

10.2.1 `title` Attribut

Mittels dieses Attributs soll der Inhalt der Seite auf prägnante Art wiedergegeben werden. Die Auswertung auf dem Datenbestand hat ergeben, dass lediglich 7 von 5000 Webseiten keine Ausprägung für dieses Attribut aufweisen. Gemäß dem intendierten Ziel der Webseitenbetreiber besitzt das Attribut eine möglichst hohe Spezifität. Es finden sich innerhalb der Stichprobe keine identischen Titel, die Merkmalsausprägung weist damit eine hohe Variabilität aus. Aufgrund dessen ist es ohne weitere Vorerarbeitungsschritte nicht für eine Gruppierung geeignet. Man könnte vor einer möglichen Gruppierung beispielsweise Methoden des Text Minings anwenden um semantisch miteinander verwandte `title` gemeinsam zu gruppieren.

10.2.2 `author` Attribut

Mittels dieses Attributs soll auf den Autor oder Urheber einer Webseite verwiesen werden. Die Auswertung auf dem Datenbestand hat ergeben, dass 4214 von 5000 Webseiten dieses Attribut nicht aufweisen oder eine leere Ausprägung für dieses Attribut aufweisen. Die Webseiten, die Ausprägungen für dieses Attribut aufweisen stammen überwiegend aus dem Mediensektor. Genannt werden meist sowohl die Namen der Autoren sowie das Medium. Es finden sich aber auch generischer Ausprägungen.

BSP: `Christiane Hoffmann, DER SPIEGEL`
`Werben & Verkaufen`

Aufgrund der insgesamt recht geringen Anzahl von Ausprägungen eignet sich das Merkmal ebenfalls weniger für einen Gruppierungsvorgang. Zudem unterscheiden sich die hier aufgeführte Inhalte semantisch recht deutlich, so dass ebenfalls zunächst eine geeignete Vorerarbeitung nötig erscheint.

10.2.3 `robots` Attribut

Wie bereits dargelegt, stellt das robots Attribut eine Indexierungsanweisung an die Crawler mit empfehlendem Charakter dar. Grundsätzlich kann ein freundlicher Webcrawler basierend auf den hier erhobenen Informationen entscheiden, ob eine Seite in seinen Suchindex aufgenommen wird. Die Auswertung auf dem Datenbestand hat ergeben, dass 870 von 5000 Webseiten dieses Attribut nicht aufweisen.

Die häufigsten Ausprägungen dieses Attributs innerhalb der Stichprobe in absteigender Reihenfolge lauten:

1. `index, follow` (1828 Ausprägungen)
2. `max-image-preview:large` (348 Ausprägungen)
3. `index, follow, max-image-preview:large, max-snippet:-1, max-video-preview:-1` (646 Ausprägungen)
4. `index, follow, noodp, noarchive, max-snippet:-1, max-image-preview:large, max-video-preview:-1` (347 Ausprägungen)

Hieraus ergibt sich eine überwiegende Mehrheit für das Folgen und Indexieren der Webseiten durch Webcrawler. Ebenfalls werden einige Informationen zur Darstellung der Vorschaugröße aus dem Bildschirm des jeweiligen Endgeräts gegeben.

Für eine Gruppierung eignet sich dieses basierend auf den Stichprobenergebnissen weniger. Überwiegend findet sich ein homogenes Verhalten wieder, nach dem die Indexierung und das Folgen gestattet wird. Immerhin 217 Webseiten besitzen demgegenüber die Anweisungen `noindex, nofollow` in Kombination und stellen damit einen Anteil von über 5% dar. Dazu kommen weitere Seiten, die jeweils eine der beiden Anweisungen aufführen.

10.2.4 `keywords` Attribut

Das `keyword` Attribut dient dazu, semantische Informationen über die Inhalte einer Webseite bereitzustellen. In den Anfängen der Suchmaschinenzeit führte eine Suchanfrage mit hoher Wahrscheinlichkeit zu der Seite, in denen möglichst alle gesuchten Begriffe vorhanden waren. Dieses Verhalten wurde infolgedessen missbraucht und findet in viele Suchmaschinen keinen Eingang mehr im positiven Sinne. Unzureichend oder zu generisch genutzte Keywords können umgekehrt aber einen negativen Effekt besitzt, damit ist das Attribut insgesamt nicht unrelevant.

Die Auswertung auf dem Datenbestand hat ergeben, dass 4195 von 5000 Webseiten und damit über 80% dieses Attribut nicht aufführen. Innerhalb der Keywordlisten finden sich Wiederholungen, die auf eine inhaltliche Verwandtschaft schließen lassen können. Häufig handelt es sich hierbei aber um sehr generische Begriffe die zudem recht volatil erscheinen, da dieses Attribut innerhalb der Stichprobe überwiegend von dem Mediensektor genutzt wird.

BSP:
```
Ukraine
    Angriff auf die Ukraine
    Geschichte
    Politik
```

Aufgrund der relativ geringen Nutzung innerhalb der Stichprobe eignet sich das Attribut nicht gut für einen Gruppierungsvorgang. Mittels einer Vorverarbeitung bzw. Zerlegung können zwar inhaltliche Verbindungen geschlossen werden, insgesamt überwiegt dabei aber der Mediensektor, da dieser das Attribut primär nutzt.

Damit würde eine Gruppierung darauf basierend insgesamt eine Voreingenommenheit erhalten. Zur Abbildung von Zusammenhängen zwischen Medien kann es dagegen eine Eignung besitzen.

10.2.5 `description` Attribut

Das `description` Attribut dient dazu, den Inhalt einer Webseite kurz darzustellen. Die hier hinterlegten Informationen werden auch in den Ergebnissen der Suchmaschinen dargestellt. Sie stellen für die SEO ein wichtiges Element dar und sollten möglichst spezifisch abgefasst sein.

Die Auswertung auf dem Datenbestand hat ergeben, dass 1408 von 5000 Webseiten dieses Attribut nicht aufführen. Unter den restlichen Ergebnissen wird erwartungsgemäß die Individualität herausgestellt, es existieren jeweils kaum Mehrfachnennungen. In einigen Fällen werden auch generische Beschreibungen wie `Sehen, verstehen, lernen` genutzt. Es überwiegt aber eine hohe Spezifität.

In der direkten Form kann dieses Attribut aufgrund der komplexen Strukturen nicht direkt für einen Gruppierungsvorgang genutzt werden. Mit einer geeigneten Vorverarbeitung könnte man die Informationen jedoch für eine inhaltliche Gruppierung nutzen. Es könnte damit die eigentliche Funktion des `keyword` Attributs übernehmen, bei einer dem `keyword` Attribut gegenüber deutlich höheren Verbreitung.

10.2.6 `content-type` Attribut

Das `content-type` Attribut dient dazu, über den in einem XML- oder HTML-Dokument verwendeten Zeichensatz zu informieren. Daher besitzt es für die korrekte Darstellung etwa im Browser Relevanz. HTML 5 nutzt anstelle dieses Attributs das Attribut `charset`.

Die Auswertung auf dem Datenbestand hat ergeben, dass alle der gezogenen 5000 Webseiten dieses Attribut nutzen. Weit über 4500 Webseiten besitzen die Merkmalsausprägung `text/html; charset=UTF-8`. In wenigen Fällen wird auch noch mittels `image/gif` oder `image/png` auf die Nutzung dieser Bildformate hingewiesen. Diese Angabe ist zwar in den stichprobenartig untersuchten Fällen hinreichend dafür, dass diese Webseiten auch Bildformate nutzen, sie stellen in der Realität aber keine Notwendigkeit dar. Falls daher erhoben werden soll, welche Formate auf einer Webseite eingebunden ist, so muss dies mittels eines expliziten Crawlvorgangs passieren.

Basierend auf den Erkenntnissen dieser Stichprobe stellt dieses Attribut kein geeignetes Gruppierungsmerkmal dar, da der Untersuchungsbereich hinsichtlich dieses Attributs bereits weitgehend homogen ist.

10.2.7 `Content-Encoding` Attribut

Das `content-encoding` Attribut besitzt analog zum Attribut `content-type` die Aufgabe, über einen verwendeten Zeichensatz zu informieren.

Die Auswertung auf dem Datenbestand hat ergeben, dass nur 8 der gezogenen 5000 Webseiten dieses Attribut nicht nutzen. Sämtliche Ausprägungen dieses Attributs innerhalb der Stichprobe in absteigender Reihenfolge lauten:

1. `UTF-8` (4911 Ausprägungen)
2. `ISO-8859-1` (76 Ausprägungen)
3. `Windows-1252` (5 Ausprägungen)

Analog zu dem Fazit der Nützlichkeit des `content-type` Attributs, ist auch dieses für eine Gruppierung nicht geeignet.

10.2.8 `Content-Language` Attribut

Das `Content-Language` Attribut verweist auf die in einem Dokument genutzte natürliche Sprache.

Die Auswertung auf dem Datenbestand hat ergeben, dass 525 der gezogenen 5000 Webseiten dieses Attribut nicht nutzen oder keine Ausprägung hinterlegen.

Sämtliche Ausprägungen dieses Attributs innerhalb der Stichprobe in absteigender Reihenfolge lauten:

1. `de` (4341 Ausprägungen)
2. `en-US` (66 Ausprägungen)
3. `en` (59 Ausprägungen)
4. `en-GB` (9 Ausprägungen)

Dabei wurden die Ausprägungen in Form verschiedener Schreibweisen harmonisiert und die einzelnen Werte entsprechend aufsummiert.

Basierend auf den Erkenntnissen dieser Stichprobe stellt dieses Attribut kein geeignetes Gruppierungsmerkmal dar, da der Untersuchungsbereich hinsichtlich dieses Attributs bereits weitgehend homogen ist. Da auch in Deutschland die Englische Sprache zunehmend genutzt wird, könnte eine zukünftige Gruppierung hinsichtlich der verwendeten Sprachen relevant werden.

10.2.9 `og:type` Attribut

Als einziges Attribut aus der von *Facebook* als API entwickeltem Open Graph Protocol, findet das Attribut `og:type` verbreitete Nutzung. Man nutzt es zur Angabe der Art des Inhalts. Beispielhafte Ausprägungen lauten `website`, `article`, `product`. Je nach Art existieren dazu auch noch genauere Ausprägungen. So kann beispielsweise die Ausprägung `video.movie` mit der Ausprägung `video:duration`, welche die Länge des Films in Sekunden spezifiziert, erweitert werden (vgl. Open Graph protocol o. D.).

Die Auswertung auf dem Datenbestand hat ergeben, dass 3554 der gezogenen 5000 Webseiten dieses Attribut nutzen.

Die häufigsten Ausprägungen dieses Attributs innerhalb der Stichprobe in absteigender Reihenfolge lauten:

1. `article` (1833 Ausprägungen)
2. `website` (1276 Ausprägungen)

Aufgrund des gewählten Startpunkts der Stichprobenerhebung ist es möglich, dass hier eine Voreingenommenheit bzgl. der Ausprägung `article` herrscht. Diese Ausprägung weißt auf die Nutzung durch den Mediensektor hin.

Basierend auf den Erkenntnissen dieser Stichprobe allein scheint dieses Attribut kein sinnvolles Merkmal für einen Gruppierungsvorgang darzustellen. Jedoch zeigt sich auch, dass verschiedene Merkmalsausprägungen auf Inhalte aus den Bereichen Sport oder auch Artikel bzw. Produkte hinweisen. Zu

untersuchen ist hier noch, ob das Protokoll einem hierarchischen Aufbau folgt oder eher eine unzusammenhängende Tag-Struktur besitzt um beispielsweise ggf. entscheiden zu können, ob und in welcher Relation verschiedene Ausprägungen zueinander stehen. Zudem ist zu bemerken, dass gegenüber dem auf der offiziellen Webseite angegebenen Beispielen die in der Stichprobe gesichteten Merkmalsausprägungen wesentlich weniger spezifisch sind.

10.2.10 `generator` Attribut

Das Attribut `generator` zeichnet das Programm aus, mit welchem eine Webseite erstellt wurde.

Die Auswertung auf dem Datenbestand hat ergeben, dass 1107 der gezogenen 5000 Webseiten dieses Attribut nutzen.

Die häufigsten Ausprägungen dieses Attributs innerhalb der Stichprobe in absteigender Reihenfolge lauten:

1. `TYPO3 CMS` (313 Ausprägungen)
2. `WordPress 5.9.2` (245 Ausprägungen)

Es finden sich unter den Ausprägungen auch wenig sinnvoll erscheinende Werte wie `search-frontend@unknown` oder `Lebenshilfe Hessen`. Diese Werte könnten ein Hinweis darauf sein, dass der eigentliche Zweck des Attributs nicht allen Beteiligten bekannt ist.

Basierend auf den Erkenntnissen dieser Stichprobe stellt dieses Attribut kein geeignetes Gruppierungsmerkmal dar, da es insgesamt wenig genutzt wird. Zudem sind die Ausprägungen auf die wichtigsten Content-Management-Systeme beschränkt, ein Rückschluss über die Nutzung auf andere für eine Gruppierung relevante Attribut erscheint daher schwierig.

10.3 Ergebnisse des Crawlvorgangs mit Breitensuche und mehreren Ausgangspunkten

Nach gegenwärtigem Stand sind 5000 zufällig gewählten Datensätze, welche mittels eines parallelen Crawlvorgangs ermittelt worden, in der Auswertungen der obigen Ergebnisse sehr ähnlich. Gegenüber den Daten fällt lediglich auch, dass die Ergebnisse des Attributs `og:type` eine größere inhaltliche Bandbreite abdecken. Dies deckt sich mit den Erwartungen, dass womöglich bei dem Crawlvorgang mit nur einem Ausgangspunkt die Stichprobengröße mit 5000 Datensätzen noch zu gering gewählt war bzw. sich dadurch inhaltlich nicht weit genug vom Ausgangspunkt entfernt wurde.

10.4 Ergebnisse des Crawlvorgangs mit Random-Walk-Strategie

Nach gegenwärtigem Stand weichen auch hier die 5000 zufällig ausgewählten Datensätze, die mittels einer Random-Walk-Strategie ausgewählt wurden, bezüglich der Auswertung der obigen Ergebnisse nicht relevant

11 Fazit / Ausblick

Die ursprünglichen Forschungsfragen bestanden darin festzustellen, welche der möglichen Metadaten von Ressourcen des WWW sich für Gruppierungen eignen. Im Folgenden wurde die Betrachtung der Ressourcen zunächst auf XML- sowie HTML-Dokumente beschränkt, bedingt durch die Entwicklung der eigenen Webcrawler. Ebenfalls wurde im Laufe der Arbeit und bei eingehender Untersuchung der Theorie und Praxis der Gruppierung bzw. Clusteranalyse deutlich, dass möglichst aussagekräftige Ergebnisse auf semantisch reichhaltigen Attributen beruhigen, von denen zumindest einige auch wenigstens metrisch skaliert seien sollten. Die Betrachtung der durch die Webcrawls gewonnenen Metadaten zeigt bezüglich der ursprünglichen Forschungsfragen ein negatives Bild auf. Zum einen existieren verschiedene semantisch reichhaltige Metadaten, die etwa aus dem Open Graph Protocol oder der Dublin Core Metadata Initiative stammen. Jedoch finden sie gegenwärtig in einem geringen Maße Anwendung, so dass mit ihnen nach aktuellem Verbreitungsstand keine Gruppierungen der Ressourcen vorgenommen werden können. Umgedreht jedoch sind die Metadaten, die weit verbreitet sind, basierend auf den gemachten Beobachtungen, überwiegend hinsichtlich des Untersuchungsgegenstand in ihren Ausprägungen entweder sehr homogen oder heterogen. In beiden Fällen kann dadurch keine sinnvolle Gruppierung vorgenommen werden. Es bleiben die semantisch reichhaltigen Schlüsselwörter und Beschreibungen, sie können nach entsprechenden Vorverarbeitungsschritten diesem Ziel dienlich sein. Es ist aber zu beachten, dass sämtliche hier erhobenen Metadaten kategorial oder maximal ordinal skaliert sind. Es existieren weitere Metadaten zu den Webseiten, etwa der Ort des Hostings oder die Besitzer, welche für Gruppierungen interessant seien, könnten, jedoch sind diese nicht direkt und allgemein einsehbar. Zudem existieren weitere auch metrisch skalierte Metadaten, wie beispielsweise die Anzahl von Bildern auf einer Webseite, diese sind aber nicht direkt mittels Apache Tika erhebbar, sondern müssen explizit im Rahmen des Crawlvorgangs erhoben werden. Zukünftige sollte man diese Informationen mit einbeziehen um zu prüfen, welche Gruppierungen basierend auf solchen Daten möglich sind.

12 Literaturverzeichnis

Balzert, Helmut/Marion Schröder/Christian Schäfer (2011): Wissenschaftliches Arbeiten, 2. Auflage, 2. Aufl., Dortmund, Deutschland: W3L GmbH.

Broder, Andrei/Ravi Kumar/Farzin Maghoul/Prabhakar Raghavan/Sridhar Rajagopalan/Raymie Stata/Andrew Tomkins/Janet Wiener (2000): Graph structure in the Web, in: Computer Networks, Bd. 33, Nr. 1–6, S. 309–320, [online] doi:10.1016/s1389-1286(00)00083-9.

Brucker, François/Jean-Pierre Barthélemy (2007): Éléments de classification - aspects combinatoires et algorithmiques, Paris, Frankreich: HERMES SCIENCE.

Clustering -- Intuition behind Kleinberg's Impossibility Theorem (2015): Cross Validated, [online] https://stats.stackexchange.com/questions/173313/clustering-intuition-behind-kleinbergs-impossibility-theorem#:%7E:text=Kleinberg%20outlines%20three%20seemingly%20intuitive,can%20satisfy%20all%20three%20simultaneously. [abgerufen am 01.05.2022].

Cornell University/Kiran Tomlinson (o. D.): Tomlinsonk / site-graph, tomlinsonk / site-graph, [online] https://github.com/tomlinsonk/site-graph/blob/master/site_graph.py [abgerufen am 09.09.2021].

Dempster, A. P./N. M. Laird/D. B. Rubin (1977): Maximum Likelihood from Incomplete Data Via theEMAlgorithm, in: Journal of the Royal Statistical Society: Series B (Methodological), Bd. 39, Nr. 1, S. 1–31, [online] doi:10.1111/j.2517-6161.1977.tb01600.x.

Donato, Debora/Stefano Leonardi/Stefano Millozzi/Panayiotis Tsaparas (2008): Mining the inner structure of the Web graph, in: Journal of Physics A: Mathematical and Theoretical, Bd. 41, Nr. 22, S. 224017, [online] doi:10.1088/1751-8113/41/22/224017.

Ester, Martin (2013): Knowledge Discovery in Databases: Techniken Und Anwendungen, 2000. Aufl., Berlin, Deutschland: Springer.

Fahrmeir, Ludwig/Christian Heumann/Rita Künstler/Iris Pigeot/Gerhard Tutz (2016): Statistik: Der Weg zur Datenanalyse (Springer-Lehrbuch), 8., überarb. u. erg. Aufl. 2016, Berlin Heidelberg, Deutschland: Springer Spektrum.

Fayyad, Usama/Cory Reina/P. S. Bradley (1998): Initialization of Iterative Refinement Clustering Algorithms, in: Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD'98), S. 194–198.

Forgy, E. W. (1965): Cluster analysis of multivariate data: Efficiency vs. interpretability of classification (abstract), in: Biometrics, Bd. 21, S. 768–769.

Gorantla, Eresh (2021): Postgres JSONB Usage and performance analysis - Geek Culture, Medium, [online] https://medium.com/geekculture/postgres-jsonb-usage-and-performance-analysis-cdbd1242a018#:%7E:text=What%20is%20JSONB%3F,string%2C%20but%20as%20binary%20code. [abgerufen am 09.03.2022].

Heydon, Allan/Najork, Marc (2004): Mercator: A scalable, extensible Web crawler, in: World Wide Web, Nr. 2, S. 219–229.

Hjørland, Birge (2017): Classification, in: Knowledge Organization, Bd. 44, Nr. 2, S. 97–128.

How to extract top-level domain name (TLD) from URL (2009): Stack Overflow, [online] https://stackoverflow.com/questions/1066933/how-to-extract-top-level-domain-name-tld-from-url$ [abgerufen am 10.04.2022].

Kaufman, Leonard/Peter J. Rousseeuw (1990): Finding Groups in Data: An Introduction to Cluster Analysis (Wiley Series in Probability and Statistics), 1., Hoboken, USA: Wiley-Interscience.

Kleinberg, Jon (2002): An Impossibility Theorem for Clustering, in: NIPS'02: Proceedings of the 15th International Conference on Neural Information Processing Systems, S. 463–470.

Leskovec, Jure/Christos Faloutsos (2006): Sampling from large graphs, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '06, S. 1–8, [online] doi:10.1145/1150402.1150479.

MacQueen, J. (1967): Some Methods for Classification and Analysis of Multivariate Observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Bd. 1, S. 281–297.

Manning, Christopher/Prabhakar Raghavan/Hinrich Schütze (2008): Introduction to Information Retrieval, Anniversary, Cambridge, England: Cambridge University Press.

Ng, Raymond T./Jiawei Han (1994): Efficient and Effective Clustering Methods for Spatial Data Mining, in: Proc. 20th Int. Conf. on Very Large Data Bases (VLDB'94), S. 144–155.

Open Graph protocol (o. D.): The Open Graph protocol, [online] https://ogp.me/#types [abgerufen am 01.05.2022].

Parrochia, Daniel (o. D.): Classification | Internet Encyclopedia of Philosophy, Internet Encyclopedia of Philosophy (IEP), [online] https://iep.utm.edu/classification-in-science/ [abgerufen am 01.05.2022].

Parrochia, Daniel (2018): Mathematical Theory of Classification, in: KNOWLEDGE ORGANIZATION, Bd. 45, Nr. 2, S. 184–201, [online] doi:10.5771/0943-7444-2018-2-184.

Sereda, Evgeni (2021): Die meistbesuchten und meistaufgerufenen Websites in Deutschland — TOP 100 in 2021 und 2022, Semrush Blog, [online] https://de.semrush.com/blog/top-der-meistbesuchten-webseiten/ [abgerufen am 01.05.2022].

Socket.timeout is thrown instead of requests.exceptions.Timeout · Issue #1236 · psf/requests (2013): GitHub, [online] https://github.com/psf/requests/issues/1236 [abgerufen am 15.04.2022].

Tomlinson, Kiran (2020): Site Graph, Site Graph - A script for generating interactive website visualizations, [online] https://www.cs.cornell.edu/%7Ekt/post/site-graph/ [abgerufen am 14.02.2022].

13 Anhang

13.1 Quellcodes der Webcrawler

Webcrawler mit implementierter Breitensuche ausgehend von einem Startpunkt:

```
import json
from bs4 import BeautifulSoup
import urllib
import requests
import tld
from tld import get_tld # um die Top-Level-Domain später auf 'de'
einzugrenzen
import socket # um Exception abzufangen, die 'requests' durchlässt
import sys  #für die Methode 'exit()' zur Programmbeendigung
import psycopg2 #für die Anbindung an das DBMS PostgreSQL
from psycopg2.extras import Json, DictCursor #ebenfalls für die Anbindung an
DBMS PostgreSQL
import os

os.environ['TIKA_SERVER_JAR'] =
'https://repo1.maven.org/maven2/org/apache/tika/tika-server/1.27/tika-
server-1.27.jar'

import tika
from tika import parser

from collections import deque

import threading


# Filtern von URLs:
def filtere_url(url_name):

    if url_name.find("?") != -1:
        url_name = url_name[: url_name.find("?") ] # URLs mit ? ab dem ?
beschneiden

    if url_name.find(".csv") != -1 or url_name.find(".pdf") != -1 or
url_name.find(".jpg") != -1 or url_name.find(".jpeg") != -1 or
url_name.find(".rtf") != -1 or url_name.find(".txt") != -1 or
url_name.find(".flac") != -1 or url_name.find(".wav") != -1 or
url_name.find(".mp3") != -1 or url_name.find(".mp4") != -1 or
url_name.find(".jp2") != -1:
        url_name = url_name[: url_name.rfind("/")]

    if url_name.find(";jsession") != -1:
        url_name = url_name[: url_name.find(";jsession") ] # falls eine
Jsession generiert wird, soll in der URL alles
                                            # ab diesem Teil abgeschnitten
werden

    if url_name.find("&sid") != -1:
```

```
        url_name = url_name[: url_name.find("&sid") ] # falls eine sid
generiert wird, soll in der URL alles
                                                # ab diesem Teil abgeschnitten
werden

    if url_name[-1:] == "/":
        url_name = url_name[:-1]  # Finales '/' abschneiden

    return url_name



# Von Klasse ableiten, um crawling thread-fähig zu machen:
class CCrawl(threading.Thread):
    def __init__(self, url, *args, **kwargs):
        super(CCrawl, self).__init__(*args, **kwargs)
        self.url = url
        self.not_visitable = set()
        self.visited = set()
        self.edges = set()
        self.html_pages = set()      # Abgrenzung zu resource_pages, die
nicht-html sind
        self.redirect_target_url = dict()


    def run(self):
        try:

            ebene = 0   # Ebene im Graph
            #ebenengroesse = 3   # Wie viele Verweise auf einer Ebene
            max_ebene = 10   # Bis Anzahl dieser Ebenen

            self.url = filtere_url(self.url)

            try:
                head = requests.head(self.url, timeout=15)
            except UnicodeError as ue:
                print("Folgender Request-Fehler: ", ue)
                return
            except requests.exceptions.RequestException as e:
                print("Folgender Request-Fehler: ", e)
                return
            except (requests.exceptions.RequestException, socket.timeout) as
e:
                print("Folgender Request-Fehler: ", e)
                return

            site_url = head.url

            site_url = filtere_url(site_url)

            self.redirect_target_url[self.url] = site_url  # falls ein
Rediret Link im HTTP Header angegeben ist,
                                                # wird er hier auch übernommen,
ansonsten bleibt die
                                                # identische URL zur Aufruf-URL
hier stehen

            to_visit = deque()
```

```
            if site_url not in self.visited and site_url not in
self.not_visitable:
                to_visit.append((site_url, None))   # an eine Double-Ended-
Queue das Tupel (site_url, None) anhängen
                                                    # entspricht dem
Startknoten

            while to_visit:
                ebenengroesse = len(to_visit)   # wie viele Elemente in
Queue?


                while ebenengroesse > 0:
                    ebenengroesse -= 1

                    url, from_url = to_visit.pop()

                    print('Visiting', url, 'from', from_url)

                    error = False
                    error_obj = None
                    try:
                        page = requests.get(url, timeout=15)
                    except UnicodeError as ue:
                        error = True
                        error_obj = ue
                    except requests.exceptions.RequestException as e:
                        error = True
                        error_obj = e
                    except (requests.exceptions.RequestException,
socket.timeout) as e:
                        error = True
                        error_obj = e

                    if error or not page or not 'html' in
page.headers.get('content-type', ''):
                        self.not_visitable.add(url)
                        continue
                    # ?? hier ggf. eine andere Behandlung nötig wegen der
Zähler etc.?
                        # denke nicht, da Ebenengroesse ja dekrementiert


                    # ?? da keinen Fehler geworfen kann sie besucht werden,
ggf. zuerst hier die
                    # relevanten Metadaten einlesen und DANACH bzw. nachdem
die Links auf dieser
                    # Seite geschürft wurden erst dann 'visited' hinzufügen
                    # -> hier direkt 'visited' hinzugefügt, dies ggf. nach
hinten verschieben
                    self.visited.add(url)
                    write_to_database_url(url)


                    soup = BeautifulSoup(page.text, 'html.parser')
                    #internal_links = set()
                    #external_links = set()

                    # Handle <base> tags
```

```
                    base_url = soup.find('base')
                    base_url = '' if base_url is None else
base_url.get('href', '')

                    for link in soup.find_all('a', href=True):
                        link_url = link['href']


                        # keine eMail-Adressen berücksichtigen
                        if link_url.startswith('mailto:'):
                            continue

                        # relative Pfade auflösen
                        if not link_url.startswith('http'):
                            link_url = urllib.parse.urljoin(site_url,
urllib.parse.urljoin(base_url, link_url))

                        # Queries bzw. Fragmente aus internen Links
entfernen
                        # Achtung: Hier habe ich link_url gegen site_url
ausgetauscht!!!
                        if link_url.startswith(site_url):
                            link_url = urllib.parse.urljoin(site_url,
urllib.parse.urlparse(link_url).path)

                        # Falls es sich um einen Redirect Link handelt, wird
sein Ziel genutzt
                        if link_url in self.redirect_target_url:
                            link_url = self.redirect_target_url[link_url]


                        link_url = filtere_url(link_url)

                        if link_url in self.visited or link_url in
self.not_visitable:
                                self.not_visitable.add(link_url)
                                continue

                        try:
                            if get_tld(link_url, fix_protocol=True) != 'de':
                                self.not_visitable.add(link_url)
                                continue
                        except:
                            self.not_visitable.add(link_url)
                            continue

                        to_visit.append((link_url, url))
                        self.edges.add((url, link_url))
                        write_to_database_edge( (url, link_url) )

                ebene += 1

                if ebene == max_ebene:
                    break


        except Exception as e:
            print('Fehler in thread: ' , e)
```

```
def write_to_database_url(in_url):
    try:
        conn = psycopg2.connect(dbname="db_sampling_one_seed_v3",
user="alexander", password="alexander", host="127.0.0.1", port="5432")
        cur = conn.cursor(cursor_factory=DictCursor)

    except:
        print("Datenbankzugriffsfehler!")


    try:
        parsed = parser.from_file(in_url)
        res = parsed["metadata"]


        cur.execute("INSERT INTO webseite (url, metadaten) VALUES (%s, %s)
ON CONFLICT (url) DO NOTHING" , [ in_url,

Json(res)

] )

        conn.commit()

    except:

        cur.execute("INSERT INTO webseite (url, metadaten) VALUES (%s, %s)
ON CONFLICT (url) DO NOTHING", [ in_url, Json("Parserfehler") ] )
        conn.commit()



def write_to_database_edge(edge):
    try:
        conn = psycopg2.connect(dbname="db_sampling_one_seed_v3",
user="alexander", password="alexander", host="127.0.0.1", port="5432")
        cur = conn.cursor(cursor_factory=DictCursor)

    except:
        print("Datenbankzugriffsfehler!")


    (org, dest) = edge

    try:
        cur.execute("INSERT INTO verlinkung (von, nach) VALUES (%s, %s) ON
CONFLICT (von, nach) DO NOTHING", [ org, dest ] )
        conn.commit()
    except Exception as e:
        print("Verlinkungsfehler: (von, nach) VALUES (%s, %s)\n", [ org,
dest ] )
        print(e, '\n')

    conn.close()
```

```
if __name__ == '__main__':

    # Sets anlegen:
    vis_set = set()
    ed_set = set()

    # Leeres Dict für späteres Updaten:
    red_dict = {}

    seeds = [
                "https://www.spiegel.de",
            ]

    threads = [CCrawl(seed) for seed in seeds]

    for t in threads:
        t.start()

    for t in threads:
        t.join()
```

Webcrawler mit implementierter Breitensuche ausgehend von mehreren Startpunkten:

```
import json
from bs4 import BeautifulSoup
import urllib
import requests
import tld
from tld import get_tld # um die Top-Level-Domain später auf 'de'
einzugrenzen
import socket # um Exception abzufangen, die 'requests' durchlässt
import sys  #für die Methode 'exit()' zur Programmbeendigung
import psycopg2 #für die Anbindung an das DBMS PostgreSQL
from psycopg2.extras import Json, DictCursor #ebenfalls für die Anbindung an
DBMS PostgreSQL
import os

os.environ['TIKA_SERVER_JAR'] =
'https://repo1.maven.org/maven2/org/apache/tika/tika-server/1.27/tika-
server-1.27.jar'

import tika
from tika import parser

from collections import deque

import threading


# Filtern von URLs:
def filtere_url(url_name):

    if url_name.find("?") != -1:
        url_name = url_name[: url_name.find("?") ]

    if url_name.find(".csv") != -1 or url_name.find(".pdf") != -1 or
url_name.find(".jpg") != -1 or url_name.find(".jpeg") != -1 or
url_name.find(".rtf") != -1 or url_name.find(".txt") != -1 or
url_name.find(".flac") != -1 or url_name.find(".wav") != -1 or
url_name.find(".mp3") != -1 or url_name.find(".mp4") != -1 or
url_name.find(".jp2") != -1:
        url_name = url_name[: url_name.rfind("/")]

    if url_name.find(";jsession") != -1:
        url_name = url_name[: url_name.find(";jsession") ] # falls eine
Jsession generiert wird, soll in der URL alles
                                            # ab diesem Teil abgeschnitten
werden
    if url_name.find("&sid") != -1:
        url_name = url_name[: url_name.find("&sid") ] # falls eine sid
generiert wird, soll in der URL alles
                                            # ab diesem Teil abgeschnitten
werden

    if url_name[-1:] == "/":
        url_name = url_name[:-1]  # Finales '/' abschneiden

    return url_name
```

```
# Von Klasse ableiten, um crawling thread-fähig zu machen:
class CCrawl(threading.Thread):
    def __init__(self, url, *args, **kwargs):
        super(CCrawl, self).__init__(*args, **kwargs)
        self.url = url
        self.not_visitable = set()
        self.visited = set()
        self.edges = set()
        self.html_pages = set()      # Abgrenzung zu resource_pages, die
nicht-html sind
        self.redirect_target_url = dict()


    def run(self):
        try:

            ebene = 0   # Ebene im Graph
            #ebenengroesse = 3   # Wie viele Verweise auf einer Ebene
            max_ebene = 10   # Bis Anzahl dieser Ebenen

            self.url = filtere_url(self.url)

            try:
                head = requests.head(self.url, timeout=15)
            except UnicodeError as ue:
                print("Folgender Request-Fehler: ", ue)
            except requests.exceptions.RequestException as e:
                print("Folgender Request-Fehler: ", e)
            except (requests.exceptions.RequestException, socket.timeout) as
e:
                print("Folgender Request-Fehler: ", e)

            site_url = head.url

            site_url = filtere_url(site_url)

            try:
                if get_tld(self.url, fix_protocol=True) != 'de' or
get_tld(site_url, fix_protocol=True) != 'de': # Prüfen auf korrekte Top-
Level-Domain
                    print("Initiale url bzw. redirect url hat keine Top-
Level-Domain 'de' :", self.url, ' ', site_url)
            except:
                print("Fehler")

            self.redirect_target_url[self.url] = site_url  # falls ein
Rediret Link im HTTP Header angegeben ist,
                                                # wird er hier auch übernommen,
ansonsten bleibt die
                                                # identische URL zur Aufruf-URL
hier stehen

                                                # ?? sollte man auch die
redirect url auf schließendes '/' prüfen?
                                                # -> so umgesetzt, schließendes
'/' abschneiden

            to_visit = deque()
```

```
            if site_url not in self.visited and site_url not in
self.not_visitable:
                to_visit.append((site_url, None))   # an eine Double-Ended-
Queue das Tupel (site_url, None) anhängen
                                                    # entspricht dem
Startknoten

            while to_visit:
                ebenengroesse = len(to_visit)   # wie viele Elemente in
Queue?


                while ebenengroesse > 0:
                    ebenengroesse -= 1

                    url, from_url = to_visit.pop()

                    print('Visiting', url, 'from', from_url)

                    error = False
                    error_obj = None
                    try:
                        page = requests.get(url, timeout=15)
                    except UnicodeError as ue:
                        error = True
                        error_obj = ue
                    except requests.exceptions.RequestException as e:
                        error = True
                        error_obj = e
                    except (requests.exceptions.RequestException,
socket.timeout) as e:
                        error = True
                        error_obj = e

                    if error or not page or not 'html' in
page.headers.get('content-type', ''):
                        self.not_visitable.add(url)
                        continue
                    # ?? hier ggf. eine andere Behandlung nötig wegen der
Zähler etc.?
                        # denke nicht, da Ebenengroesse ja dekrementiert


                    # ?? da keinen Fehler geworfen kann sie besucht werden,
ggf. zuerst hier die
                    # relevanten Metadaten einlesen und DANACH bzw. nachdem
die Links auf dieser
                    # Seite geschürft wurden erst dann 'visited' hinzufügen
                    # -> hier direkt 'visited' hinzugefügt, dies ggf. nach
hinten verschieben
                    self.visited.add(url)
                    write_to_database_url(url)


                    soup = BeautifulSoup(page.text, 'html.parser')

                    # Handle <base> tags
                    base_url = soup.find('base')
                    base_url = '' if base_url is None else
base_url.get('href', '')
```

```
                    for link in soup.find_all('a', href=True):
                        link_url = link['href']


                        # keine eMail-Adressen berücksichtigen
                        if link_url.startswith('mailto:'):
                            continue

                        # relative Pfade auflösen
                        if not link_url.startswith('http'):
                            link_url = urllib.parse.urljoin(site_url,
urllib.parse.urljoin(base_url, link_url))

                        # Queries bzw. Fragmente aus internen Links
entfernen
                        if link_url.startswith(site_url):
                            link_url = urllib.parse.urljoin(link_url,
urllib.parse.urlparse(link_url).path)

                        # Falls es sich um einen Redirect Link handelt, wird
sein Ziel genutzt
                        if link_url in self.redirect_target_url:
                            link_url = self.redirect_target_url[link_url]


                        link_url = filtere_url(link_url)


                        if link_url in self.visited or link_url in
self.not_visitable:
                            self.not_visitable.add(link_url)
                            continue

                        try:
                            if get_tld(link_url, fix_protocol=True) != 'de':
                                continue
                        except:
                            self.not_visitable.add(link_url)
                            continue

                        to_visit.append((link_url, url))
                        self.edges.add((url, link_url))
                        write_to_database_edge( (url, link_url) )

                ebene += 1

                if ebene == max_ebene:
                    break


        except Exception as e:
            print('Fehler in thread: ' , e)
            return



def write_to_database_url(in_url):
    try:
```

```
        conn = psycopg2.connect(dbname="db_sampling_many_seeds",
user="alexander", password="alexander", host="127.0.0.1", port="5432")
        cur = conn.cursor(cursor_factory=DictCursor)

    except:
        print("Datenbankzugriffsfehler!")


    try:
        parsed = parser.from_file(in_url)
        res = parsed["metadata"]

        cur.execute("INSERT INTO webseite (url, metadaten) VALUES (%s, %s)
ON CONFLICT (url) DO NOTHING", [ in_url,

Json(res)

] )
        conn.commit()

    except:

        cur.execute("INSERT INTO webseite (url, metadaten) VALUES (%s, %s)",
[ in_url, Json("Parserfehler") ] )
        conn.commit()

    conn.close()


def write_to_database_edge(edge):
    try:
        conn = psycopg2.connect(dbname="db_sampling_many_seeds",
user="alexander", password="alexander", host="127.0.0.1", port="5432")
        cur = conn.cursor(cursor_factory=DictCursor)

    except:
        print("Datenbankzugriffsfehler!")


    (org, dest) = edge

    try:
        cur.execute("INSERT INTO verlinkung (von, nach) VALUES (%s, %s) ON
CONFLICT (von, nach) DO NOTHING", [ org, dest ] )
        conn.commit()
    except Exception as e:
        print("Verlinkungsfehler: (von, nach) VALUES (%s, %s)\n", [ org,
dest ] )
        print(e, '\n')

    conn.close()




if __name__ == '__main__':

    # Sets anlegen:
```

```
vis_set = set()
ed_set = set()

# Leeres Dict für späteres Updaten:
red_dict = {}

seeds = [
            "https://www.spiegel.de",
            "https://www.amazon.de",
            "https://www.wikipedia.de",
            "https://www.kicker.de",
            "https://www.wunderweib.de",
            "https://www.bunte.de",
            "https://www.br.de",
            "https://www.markt.de",
            "https://www.rnd.de",
            "https://www.fr.de",
            "https://www.gala.de",
            "https://www.check24.de",
            "https://www.tvspielfilm.de",
            "https://www.transfermarkt.de",
            "https://www.sueddeutsche.de",
            "https://www.wetter.de",
            "https://www.ebay.de",
            "https://www.ebay-kleinanzeigen.de",
            "https://www.stern.de",
            "https://www.telekom.de",
            "https://www.t-online.de",
            "https://www.focus.de",
            "https://www.dhl.de",
            "https://www.otto.de",
            "https://www.derwesten.de",
            "https://www.merkur.de",
            "https://www.wetteronline.de",
            "https://www.chefkoch.de",
            "https://www.sport1.de",
            "https://www.idealo.de",
            "https://www.zdf.de",
            "https://www.zeit.de",
            "https://www.bahn.de",
            "https://www.ndr.de",
            "https://www.mobile.de",
            "https://www.stern.de",
            "https://www.kaufland.de",
            "https://www.zeit.de",
            "https://www.tum.de",
            "https://www.tagesschau.de",
            "https://www.n-tv.de",
            "https://www.welt.de",
            "https://www.chip.de",
            "https://www.heise.de",
            "https://www.rtl.de",
            "https://www.giga.de",
            "https://www.mediamarkt.de",
        ]

threads = [CCrawl(seed) for seed in seeds]

for t in threads:
    t.start()
```

```
for t in threads:
    t.join()
```

Webcrawler mit implementierter Random-Walk-Strategie

```
import random # (Pseudo)Zufallszahlengenerator
import json
from bs4 import BeautifulSoup
import urllib
import requests
import tld
from tld import get_tld # um die Top-Level-Domain später auf 'de'
einzugrenzen
import socket # um Exception abzufangen, die 'requests' durchlässt
import sys  #für die Methode 'exit()' zur Programmbeendigung
import psycopg2 #für die Anbindung an das DBMS PostgreSQL
from psycopg2.extras import Json, DictCursor #ebenfalls für die Anbindung an
DBMS PostgreSQL
import os

os.environ['TIKA_SERVER_JAR'] =
'https://repo1.maven.org/maven2/org/apache/tika/tika-server/1.27/tika-
server-1.27.jar'

import tika
from tika import parser

from collections import deque


# Filtern von URLs:
def filtere_url(url_name):

   if url_name.find("?") != -1:
        url_name = url_name[: url_name.find("?") ] # URLs mit ? ab dem ?
beschneiden

    if url_name.find(".csv") != -1 or url_name.find(".pdf") != -1 or
url_name.find(".jpg") != -1 or url_name.find(".jpeg") != -1 or
url_name.find(".rtf") != -1 or url_name.find(".txt") != -1 or
url_name.find(".flac") != -1 or url_name.find(".wav") != -1 or
url_name.find(".mp3") != -1 or url_name.find(".mp4") != -1 or
url_name.find(".jp2") != -1:
        url_name = url_name[: url_name.rfind("/")]

    if url_name.find(";jsession") != -1:
        url_name = url_name[: url_name.find(";jsession") ] # falls eine
Jsession generiert wird, soll in der URL alles
                                            # ab diesem Teil abgeschnitten
werden

    if url_name.find("&sid") != -1:
        url_name = url_name[: url_name.find("&sid") ] # falls eine sid
generiert wird, soll in der URL alles
                                            # ab diesem Teil abgeschnitten
werden

    if url_name[-1:] == "/":
        url_name = url_name[:-1]  # Finales '/' abschneiden

    return url_name
```

```
def rw_crawl(start_url):
    not_visitable = set()
    visited = set()
    edges = set()
    redirect_target_url = dict()
    all_new_discovered_urls = set()

    max_html_pages = 5000   # Gehe Irrweg bis diese Anzahl von Ressourcen

    random.seed() # Zufallszahlengenerator mit Systemzeit parametrisieren

    try:
        page = requests.get(start_url, timeout=15)
    except UnicodeError as ue:
        print("INITIALER Get-Fehler: ", ue)
        return visited, edges, resource_pages, redirect_target_url
    except requests.exceptions.RequestException as e:
        print("INITIALER Get-Fehler: ", e)
        return visited, edges, resource_pages, redirect_target_url
    except (requests.exceptions.RequestException, socket.timeout) as e:
        print("INITIALER Get-Fehler: ", e)
        return visited, edges, resource_pages, redirect_target_url

    site_url = page.url
    site_url = filtere_url(site_url)

    redirect_target_url[start_url] = site_url  # falls ein Redirect Link im
HTTP Header angegeben ist,
                                    # wird er hier auch übernommen,
ansonsten bleibt die
                                    # identische URL zur Aufruf-URL hier
stehen

    akt_html_pages = 1
    visited.add(start_url)
    visited.add(site_url)

    # implizite Annahme, dass 1. Ressource HTML-Seite ist und keine Fehler
produziert mit korrekter Top-Level-Domain


    while akt_html_pages <= max_html_pages:

        soup = BeautifulSoup(page.text, 'html.parser')


        # Handle <base> tags
        base_url = soup.find('base')
        base_url = '' if base_url is None else base_url.get('href', '')

        resource_pages_out_links = set()


        # hier alternatives Vorgehen, falls gar kein Link da
        for link in soup.find_all('a', href=True):
            link_url = link['href']
```

```
            # keine eMail-Adressen berücksichtigen
            if link_url.startswith('mailto:'):
                continue

            # relative Pfade auflösen
            if not link_url.startswith('http'):
                link_url = urllib.parse.urljoin(site_url,
urllib.parse.urljoin(base_url, link_url))

            # Queries bzw. Fragmente aus internen Links entfernen
            if link_url.startswith(site_url):
                link_url = urllib.parse.urljoin(link_url,
urllib.parse.urlparse(link_url).path)

            link_url = filtere_url(link_url)


            if link_url not in not_visitable:
                resource_pages_out_links.add(link_url)
                all_new_discovered_urls.add(link_url)

        if len( resource_pages_out_links ) == 0:
            error = True
            while error:
                next_link = random.choice( list(all_new_discovered_urls) )

                try:
                    page = requests.get(next_link, timeout=15)
                    if page and 'html' in page.headers.get('content-type',
''):
                        is_html = True
                except UnicodeError as ue:
                    error = True
                    error_obj = ue
                except requests.exceptions.RequestException as e:
                    error = True
                    error_obj = e
                except (requests.exceptions.RequestException,
socket.timeout) as e:
                    error = True
                    error_obj = e

                if error or not page or not is_html:
                    error = True
                    not_visitable.add(next_link)
                    all_new_discovered_urls =
all_new_discovered_urls.difference( { next_link } )

                    next_link = random.choice( list(all_new_discovered_urls)
)

                else:
                    error = False
            continue

        else:
            next_link = random.choice( list(resource_pages_out_links) )

        error = True
```

```
        while error:

            is_html = False
            error = False
            error_obj = None

            # Hier erst prüfen, ob überhaupt Element gezogen wurde ->
ansonsten ist Menge leer...keine Out-Links mehr

            try:
                page = requests.get(next_link, timeout=15)
                if page and 'html' in page.headers.get('content-type', ''):
                    is_html = True
            except UnicodeError as ue:
                error = True
                error_obj = ue
            except requests.exceptions.RequestException as e:
                error = True
                error_obj = e
            except (requests.exceptions.RequestException, socket.timeout) as
e:
                error = True
                error_obj = e

            if error or not page or not is_html:
                error = True
                not_visitable.add(next_link)
                resource_pages_out_links =
resource_pages_out_links.difference( { next_link } )

                next_link = random.choice( list(resource_pages_out_links) )

            else:
                error = False

                edges.add( (site_url, filtere_url(page.url) ) )
                write_to_database_edge( (site_url, filtere_url(page.url) ) )
                visited.add(filtere_url(page.url))
                write_to_database_url(filtere_url(page.url))

                redirect_target_url[next_link] = filtere_url(page.url) #
falls ein Redirect Link im HTTP Header angegeben ist,

                visited.add(next_link)
                visited.add(filtere_url(page.url))


            akt_html_pages += 1

            print('Visiting', next_link, 'from', site_url)
            site_url = filtere_url(page.url)

    return



def write_to_database_url(in_url):
    try:
        conn = psycopg2.connect(dbname="db_sampling_one_seed_v3",
user="alexander", password="alexander", host="127.0.0.1", port="5432")
```

```
        cur = conn.cursor(cursor_factory=DictCursor)

    except:
        print("Datenbankzugriffsfehler!")


    try:
        parsed = parser.from_file(in_url)
        res = parsed["metadata"]


        cur.execute("INSERT INTO webseite (url, metadaten) VALUES (%s, %s)
ON CONFLICT (url) DO NOTHING" , [ in_url,

Json(res)

] )

        conn.commit()

    except:

        cur.execute("INSERT INTO webseite (url, metadaten) VALUES (%s, %s)
ON CONFLICT (url) DO NOTHING", [ in_url, Json("Parserfehler") ] )
        conn.commit()



def write_to_database_edge(edge):
    try:
        conn = psycopg2.connect(dbname="db_sampling_one_seed_v3",
user="alexander", password="alexander", host="127.0.0.1", port="5432")
        cur = conn.cursor(cursor_factory=DictCursor)

    except:
        print("Datenbankzugriffsfehler!")


    (org, dest) = edge

    try:
        cur.execute("INSERT INTO verlinkung (von, nach) VALUES (%s, %s) ON
CONFLICT (von, nach) DO NOTHING", [ org, dest ] )
        conn.commit()
    except Exception as e:
        print("Verlinkungsfehler: (von, nach) VALUES (%s, %s)\n", [ org,
dest ] )
        print(e, '\n')

    conn.close()



if __name__ == '__main__':


    seeds = [
                "https://www.spiegel.de",
            ]
```

```
threads = [CCrawl(seed) for seed in seeds]

for t in threads:
    t.start()

for t in threads:
    t.join()
```

13.2 Analyse der Daten

Analyse der Metadaten:

```
# Temporär Sprache auf Englisch umstellen, um die Fehlermeldungen im
Original zu sehen:
Sys.setlocale("LC_MESSAGES", "en_US.utf8")

install.packages('sqldf')
install.packages('RPostgreSQL')
install.packages('jsonlite')
library('DBI')
library('RPostgreSQL')
library('jsonlite')
library('dplyr')

# create a connection object:
conn <- dbConnect(drv=PostgreSQL(),
                  user="alexander",
                  password="alexander",
                  host="localhost",
                  port=5432,
                  dbname="db_scraping_results")

dbListTables(conn)

# Statistik mit 5000 zufälligen Datensätzen aus der geschürften Menge:
webseiten <- dbGetQuery(conn, "SELECT * FROM webseite ORDER BY random();")

# rausfiltern der Einträge "Parserfehler" in Spalte metadaten:
webseiten <- webseiten[webseiten$metadaten != '"Parserfehler"',]

# Untersuchung zunächst normieren auf 5000 Datensätze:
webseiten <- webseiten[1:5000, ]

# Metadaten in DataFrame umwandeln
metadaten <- jsonlite::stream_in(textConnection(gsub("\\n", "",
webseiten$metadaten)))

# Spalte id hinten angehängt um spätere Zuordnung sicherzustellen
webseiten_ids <- webseiten$id
metadaten['id'] <- c(webseiten_ids)

webseiten_mit_metadaten <- merge(webseiten, metadaten, by="id")


# String NULL durch echtes NA in R ersetzen:
webseiten_mit_metadaten[webseiten_mit_metadaten == "NULL"]<-NA

# String NA durch echtes NA in R ersetzen:
webseiten_mit_metadaten[webseiten_mit_metadaten == "NA"]<-NA


# Spalte metadaten entfernen:
webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(metadaten))
```

```
# zu untersuchende Attribute:
webseiten_mit_metadaten <- subset(webseiten_mit_metadaten,
                                  select = c(id,
                                             url.x,
                                             title,
                                             author,
                                             robots,
                                             Robots,
                                             ROBOTS,
                                             keywords,
                                             Keywords,
                                             copyright,
                                             description,
                                             `Content-Type`,
                                             `Content-Encoding`,
                                             `Content-Language`,
                                             `content-language`,
                                             `Content-language`,
                                             `og:type`,
                                             generator,
                                             Generator))


# Metadaten harmonisieren: Spalten mit unterschiedlichen Schreibweisen aber
identischem
# Inhalt angleichen, so dass nur noch jeweils eine Spalte existiert:

webseiten_mit_metadaten$robots <-
ifelse(is.na(webseiten_mit_metadaten$robots),
                                          webseiten_mit_metadaten$Robots,
                                          webseiten_mit_metadaten$robots)

webseiten_mit_metadaten$robots <-
ifelse(is.na(webseiten_mit_metadaten$robots),
                                          webseiten_mit_metadaten$ROBOTS,
                                          webseiten_mit_metadaten$robots)

webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(Robots))
webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(ROBOTS))


webseiten_mit_metadaten$keywords <-
ifelse(is.na(webseiten_mit_metadaten$keywords),

webseiten_mit_metadaten$Keywords,

webseiten_mit_metadaten$keywords)

webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(Keywords))

webseiten_mit_metadaten$`Content-Language` <-
ifelse(is.na(webseiten_mit_metadaten$`Content-Language`),

webseiten_mit_metadaten$`content-language`,

webseiten_mit_metadaten$`Content-Language`)
```

```
webseiten_mit_metadaten$`Content-Language` <-
ifelse(is.na(webseiten_mit_metadaten$`Content-Language`),

webseiten_mit_metadaten$`Content-language`,

webseiten_mit_metadaten$`Content-Language`)

webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(`content-language`))
webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(`Content-language`))

webseiten_mit_metadaten$generator <-
ifelse(is.na(webseiten_mit_metadaten$generator),

webseiten_mit_metadaten$Generator,

webseiten_mit_metadaten$generator)

webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(`Generator`))

webseiten_mit_metadaten$language <-
ifelse(is.na(webseiten_mit_metadaten$language),

webseiten_mit_metadaten$Language,

webseiten_mit_metadaten$language)

webseiten_mit_metadaten$language <-
ifelse(is.na(webseiten_mit_metadaten$language),

webseiten_mit_metadaten$lang,

webseiten_mit_metadaten$language)

webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(Language))
webseiten_mit_metadaten <- subset(webseiten_mit_metadaten, select = -
c(lang))


# Beginn der Analyse:


# title:

  #Anzahl Seiten ohne Titel:
sum(is.na(webseiten_mit_metadaten$title))

  #Zudem sehr hohe Variabilität (nicht Varianz, da dies def. ist mit
Standardabweichung)
webseiten_mit_metadaten %>% count(title)


# author:

#Anzahl Seiten ohne author:
sum(is.na(webseiten_mit_metadaten$author))
```

```
# besser als 'sum', da hier auch leere Werte extra ausgewiesen werden:
(Hinweis: NA und leerstring zusammen) )
webseiten_mit_metadaten %>% count(author)

table( unlist(webseiten_mit_metadaten$author), useNA = "ifany")


# robots:
webseiten_mit_metadaten %>% count(robots)


# keywords:

sum(is.na(webseiten_mit_metadaten$keywords))
webseiten_mit_metadaten %>% count(keywords)


#  copyright:
table( webseiten_mit_metadaten$copyright )

webseiten_mit_metadaten %>% count(copyright)

# description:

sum(is.na(webseiten_mit_metadaten$description))

table( unlist(webseiten_mit_metadaten$description ) )

webseiten_mit_metadaten %>% count(description)

# Content-Type:

sum(is.na(webseiten_mit_metadaten$`Content-Type`))
webseiten_mit_metadaten %>% count(`Content-Type`)


# Content-Encoding:

sum(is.na(webseiten_mit_metadaten$`Content-Encoding`))
webseiten_mit_metadaten %>% count(`Content-Encoding`)


# Content-Language:

sum(is.na(webseiten_mit_metadaten$`Content-Language`))
webseiten_mit_metadaten %>% count(`Content-Language`)


# og:type:

sum(is.na(webseiten_mit_metadaten$`og:type`))
webseiten_mit_metadaten %>% count(`og:type`)


# generator:

sum(is.na(webseiten_mit_metadaten$generator))
webseiten_mit_metadaten %>% count(`generator`)
```

```
# language:

sum(is.na(webseiten_mit_metadaten$language))
webseiten_mit_metadaten %>% count(`language`)
```

Erklärung

Hiermit bestätige ich, dass ich die wissenschaftliche Arbeit mit dem Titel *Untersuchung von Teilbereichen des Internets im Hinblick auf mögliche Gruppierungen basierend auf diskreten Merkmalen* selbstständig und ohne fremde Hilfe angefertigt habe. Alle Stellen, welche wortwörtlich oder sinngemäß zitiert oder übernommen wurden, sind auch als solche kenntlich gemacht.

Des Weiteren bestätige ich, dass das Thema meiner wissenschaftlichen Arbeit von mir stammt und nicht von einer schon vorhandenen Arbeit übernommen wurde. Diese Arbeit wurde noch keiner Prüfungsbehörde vorgelegt und nicht veröffentlicht.

Name, Vorname: Mengel, Alexander
Matrikelnummer: 70453431
Ort, Datum: Augsburg, 21. Mai 2022
Unterschrift: