Konzeption und Erprobung eines Webcrawlers zur Erstellung hierarchischer Indizes

Mathias Haimerl
evosoft GmbH, Business Function Application & Platform Solutions
Nürnberg
mathias.haimerl@evosoft.com

ZUSAMMENFASSUNG

Das Durchsuchen von Webseiten, wie es u.A. von modernen *Topical Crawlers* (vgl. [16]) betrieben wird, ist technisch äußerst aufwändig, da der Fokus auf Extraktion und Korrelation sämtlicher Informationen einer Webseite liegt. Für die Markierung und computergestützte Erklärung komplexer Ausdrücke in Texten müssen diese zuvor identifiziert und indiziert werden. Um eine Webseite auf Basis eines bestehenden *Grundindex* zu durchsuchen und einen für die Einzelseite spezifischen *Subindex* zu erstellen, muss ein alternatives Konzept des *Crawlings* verfolgt werden, um dieses Verfahren effizient und zielgerichtet nutzen zu können. Anschließend wird die Implementierung des Crawlers skizziert und Testläufe an verschiedenartigen Webseiten getestet. Abschließend wird das erstellte Programm im Vergleich zu Topical Crawlers und der potentiellen Einsetzbarkeit im angedachten Einsatzzweck betrachtet.

CCS CONCEPTS

• **Social and professional topics** → **People with disabilities**; • **Theory of computation** → **Data structures and algorithms for data management**; • **Information systems** → *Specialized information retrieval*.

KEYWORDS

UXD, Webcrawler, Barrierefreiheit, Ordiphrase

1 MOTIVATION & BEGRIFFSERKLÄRUNG

Webcrawler oder *Spider* sind das Kernstück jeder Suchmaschine. Je nach konkreter Technologie werden Webseiten durchsucht, die Inhalte indiziert und Verknüpfungen auf andere Seiten gesammelt, um diese zu einem späteren Zeitpunkt zu besuchen. Dabei werden Schlagworte automatisch korreliert, die auf Seiten zusammen vorhanden sind und Gewichtungen aufgrund diverser Metriken erstellt [15]. Das Ziel der vollständigen Erfassung aller Informationen ist nicht für alle Anwendungsfälle notwendig, geschweige denn sinnvoll. Im Projekt *Ordiphrase* (vorgestellt in [7]) wird die Idee einer Vereinfachung der textuellen Inhalte auf Basis einer durch Crowdsourcing erfassten Datenbank an Begriffen und deren Beschreibung in einfacher Sprache vorgestellt (Definition und Abgrenzung zu *Leichter Sprache* unter [9]). Damit die Ausdrücke bei Anzeige der Webseite im Browser des Besuchers schnell übersetzt werden, darf nicht die gesamte Datenbank an Ausdrücken G geladen werden, sondern nur die Untermenge $P \subset G$, die tatsächlich in den Textknoten auf der Seite enthalten ist. Dadurch muss vor dem Laden der Webseite durch den Besucher diese Untermenge bereits bekannt sein, was bedeutet, dass asynchron zu den Seitenaufrufen durch Nutzer ein vorausgehendes Durchsuchen der Webseite unerlässlich ist. Dies genau ist die zentrale Aufgabe eines Crawlers, wobei der Fokus, wie eingangs beschrieben, anders liegt und daher eine völlig andere Verarbeitung der Daten angewandt werden muss.

2 ABGRENZUNG UND KONKRETE ZIELSETZUNG

Das Niveau der Texte ist ausschlaggebend für die Anwendbarkeit einer ausdrucksweisen Übersetzung. Für komplexe fachlich gebundene oder Texte in „Behördensprache“, kann die Erklärung einzelner Ausdrücke nicht den kompletten Kontext erschließen, da ein tief gehendes Fachwissen vorausgesetzt wird. Dieses kann durch technische Hilfsmittel einem Leser nicht einfach mitgegeben werden. Dadurch beschränkt sich die Zielsetzung dieses Projektes auf Texte des täglichen Lebens, wie Nachrichten bzw. Beiträge auf Weblogs. Dadurch wird im Rahmen der Verbesserung der Inklusion ganz bewusst nicht auf komplexe Themen fokussiert, die auch der durchschnittliche Leser nicht ohne eingehende Einarbeitung vollständig verstehen kann, sondern auf für die Allgemeinheit gedachte Texte.

MuC'19 Workshops, Hamburg, Deutschland

https://doi.org/10.18420/muc2019-ws-550

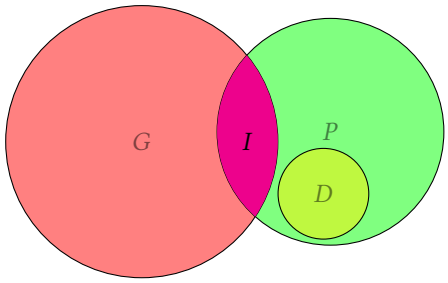

Abbildung 1: Darstellung der Indizes G und I und der Menge der Ausdrücke auf der Seite P mit mehrfach auf der Seite auftretenden Ausdrücken D als Mengendiagramm

3 MATHEMATISCHE BETRACHTUNG

Zur Entwicklung eines geeigneten Algorithmus werden die Wörter in der Seite als eine Menge von Ausdrücken P dargestellt. Dabei ist ein Ausdruck eine Menge von $1 \ldots n$ Wörtern, getrennt von einem Leerzeichen (*Space*; ASCII $0x20$), wobei dadurch der Beginn eines neuen möglichen Ausdrucks impliziert werden kann, jedoch nicht das Ende eines Ausdrucks zu bedeuten hat. Wie viele Worte ein Ausdruck umfasst, wird durch diesen selbst festgelegt; es gibt keine konzeptionelle Beschränkung. Im Gegensatz zu klassischen Crawlern können Bindewörter in diesem Algorithmus nicht entfernt werden, da sie Teil eines Ausdrucks sein können (z.B. *„Büchse der Pandora“*) und somit durch das Entfernen der Bindewörter Ausdrücke auf der Seite von der Suche nicht erfasst werden können.
Ähnlich verhält es sich mit Ausdrucks-Dubletten D: wird ein Ausdruck im Text entdeckt, der bereits zuvor gefunden wurde, kann es sein, dass dieses Vorkommen ein Teil eines anderen, aus weiteren Worten bestehenden Ausdrucks ist. Daher muss die Suche in jedem Fall fortgesetzt werden. Lediglich bei der Erstellung des konkreten Seitenindex I können tatsächlich als Dubletten identifizierte Ausdrücke nur als ein Eintrag gespeichert werden. Daraus ergibt sich für einen Seitenindex I die Definition:

$$I = G \cap (P \setminus D) \tag{1}$$

Der Zusammenhang zwischen den beteiligten Mengen wurde in Abb. 1 grafisch dargestellt. Weiterhin folgt hieraus, dass $|I| \ll |G|$ und $|I| < |P|$, wodurch die Menge der auf der Webseite vorhandenen Ausdrücke kleinstmöglich dargestellt werden kann. Beim Aufruf der Seite zu einem späteren Zeitpunkt kann I vom Server geladen werden und wird im Browser des Besuchers auf P abgebildet. Diese Funktion $t : I \longrightarrow P$ kann allgemein mit dem Aufruf der Seite ausgeliefert werden und die Ausdrücke on-demand abrufen.

4 KONSTRUKTION DES SUCHALGORITHMUS

Durch die Möglichkeit, dass Ausdrücke aus mehreren Worten bestehen, kann der Inhalt einer Webseite nicht einfach von einem *Tokenizer/Lexer* in Worte oder Teilsätze zerlegt und anschließend einzeln analysiert werden [12, 13], sondern der Text als gesamter Informationsträger muss, beginnend nach jedem Worttrennzeichen, nach bekannten Ausdrücken durchsucht werden. Das bedeutet auch, dass der Crawler bei der Suche die vollständige Sammlung aller erfassten Ausdrücke G in einem lokalen Speicher, bestenfalls im Hauptspeicher halten muss. Damit die Suche von Ausdrücken performant ist und die Belastung des ausführenden Systems niedrig bleibt, muss eine optimale Datenstruktur für das Halten im Arbeitsspeicher genutzt werden. Beim zeichenweisen Durchsuchen eines Texts gibt es für jeden Schritt 3 Möglichkeiten:

(1) Es wurde für den Text zur aktuellen Position kein Ergebnis gefunden. Im nächsten Schritt, also ab dem nächsten Zeichen nach einem Worttrennzeichen muss die Suche von vorne begonnen werden.

(2) Für das nächste Zeichen gibt es mehrere Ausdrücke, die von ihrem jeweiligen Beginn bis zum aktuellen Zeichen identisch sind. Also muss für das nächste Zeichen eine weitere Verfeinerung der möglichen Ergebnisse vorgenommen werden.

(3) Für das Zeichen gibt es nur noch eine im System erfasste Möglichkeit. Das bedeutet, dass getestet werden muss, ob der gefundene Ausdruck vollständig im Text vorkommt oder ein falsch-positives Ergebnis gefunden wurde.

Insbesondere die Nachbehandlung der gefundenen Ausdrücke ist essentiell für die korrekte Erstellung des Index. Folgendes Beispiel sollte das verdeutlichen: Im System sind die Ausdrücke *[„rotation“, „rotarmist“]* erfasst. Im Text ist der Ausdruck *„rotarier“* vorhanden. Somit werden bei der Suche bis zur Teilzeichenkette *„rota“* beide Ausdrücke als potentiell gültig bewertet. Beim nächsten Zeichen wird für die Zeichenkette *„rotar“* nur noch der Ausdruck *„rotarmist“* als potentieller Treffer gefunden. Um zu verifizieren, ob der potentielle Ausdruck tatsächlich der im Text vorkommende ist, muss ab Suchbeginn der Text mit dem gefundenen verglichen werden. Im vorliegenden Beispiel wird getestet ob der Text *„rotarier“* das potentielle Ergebnis *„rotarmist“* enthält. Da dieser Vergleich fehlschlägt, ist bewiesen, dass der Text an dieser Stelle keinen erfassten Ausdruck enthält.
Diese Suche lässt sich hervorragend mit einer Baumstruktur für die erfassten Ausdrücke als Lexikon implementieren, bei dem diese nach ihren Anfangsbuchstaben in einen Baum einsortiert werden, wobei gemeinsame Anfangssequenzen

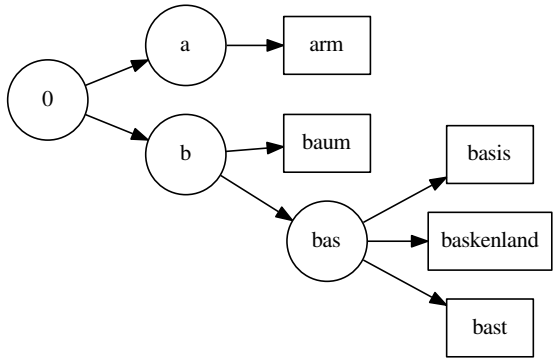

Abbildung 2: Baum des globalen Index zum Durchsuchen. Die Zweige können durchsucht werden, jedes Blatt ist ein erfasster Ausdruck.

in einem weiteren Zweig zusammengefasst werden. Durch die Verwendung beliebiger Buchstaben für die Identifikation des nachfolgenden Knotens eignet sich für diesen Anwendungsfall ein *Rosenbaum* oder *Multiway Tree* am besten [10, S. 481-489]. Die so entstehende Struktur ist exemplarisch in Abb. 2 dargestellt. Dieser Baum kann, einmal aufgebaut, allen Suchprozessen zentral zur Verfügung gestellt werden. Als weitere Optimierungsmöglichkeit zur Vermeidung von falsch-negativen Ergebnissen, also eines fehlenden Erfassens einzelner Ausdrücke müssen die Metainformationen bei Substantiven enthalten, dass es sich bei dem Ausdruck um ein Substantiv oder Eigennamen handelt. Dadurch wird eine Suche hierbei unter Beachtung der Großschreibung durchgeführt, in allen anderen Fällen ohne diese.
Wird die Suche gestartet, muss zunächst jeder mögliche Ansatzpunkt eines Ausdrucks gefunden werden. Da jeder Ausdruck am Anfang eines Worts beginnen kann, muss die Suche am Beginn jedes Wortes begonnen werden. Diese Ansatzpunkte lassen sich einfach durch das vorhergehende Durchsuchen des Textes S nach nicht-alphanumerischen Zeichen finden. Ansatzpunkte sind der Beginn des Textes $s_0 = S[0]$ und jedes Zeichen $s_n \in S$ für das gilt:

$$s_n = S[n] \forall n : \neg \,\text{alnum}(S[n-1]) \wedge \text{alnum}(S[n]) \qquad (2)$$

Wenn in einem ersten Schritt alle Ansatzpunkte $s_{0\ldots n}$ im Text gefunden wurden, kann die Suche mit der Basis s in parallelen Threads gestartet werden, da die Suchprozesse voneinander unabhängig sind. Der Ablauf eines einzelnen Threads ist nachfolgend in Algorithmus 1 dargestellt.

```
Def: phrase_t extends branch_t;
Input: branch_t*< Immutable > G := Baum aller
       erfasster Ausdrücke
Output: phrase_t*< Synchronized > I := Inital leere
        Liste von Ausdrücken
begin
    char* P_off := Einzelner Text der Webseite,
     beginnend nach einem Trennzeichen;
    size_t curTokenLen := 0;
    branch_t* curSubTree := G;
    for c ∈ P do
        curSubTree := findInSubTree(curSubTree, c,
         curTokenLen);
        if curSubTree == NULL then
            /* Kein Treffer */ break;
        end
        if curSubTree instanceof branch_t then
            /* Mehrere Ergebnisse */ curTokenLen +=
             curSubTree.steps;
            continue(curSubTree.steps);
        end
        if curSubToken instanceof phrase_t then
            /* Potentielles Ergebnis */ if checkTextRest(P,
             curSubToken) then
                /* Verifizierter Treffer */
                I.push(curSubTree); break;
            end
        end
    end
end
```

Algorithm 1: Algorithmus für einen einzelnen Thread der Suche.

5 INDIZIERUNG DER SUCHBASIS

Ziel ist es, dem Betreiber einer Webseite nahezu ohne eigenes Zutun die volle Funktionalität bereitzustellen. Dafür kann ein Betreiber nach der Registrierung angeben, welche Domains seine Webinhalte bereitstellen und indiziert werden sollen. Da jede Website aus einer Vielzahl an Unterseiten besteht, sollen diese automatisch erfasst und nicht zusätzlich verwaltet werden müssen. Daher muss eine Vorbereitung vor dem Durchsuchen der Einzelseiten selbstständig alle diese ermitteln. Es gibt hierfür mehrere Möglichkeiten, deren Aufwand sehr unterschiedlich ist und die zusätzlich zum Teil weitere Informationen nutzen:

(1) Stellt die Webseite eine `sitemap.xml` zur Verfügung, so ist es ausreichend, die in dieser Datei - oder weiterer darin verlinkter Dateien - zu durchsuchen, da der Betreiber des Webauftritts hier alle Seiten, auf die Besucher stoßen sollen, eigenverantwortlich eintragen

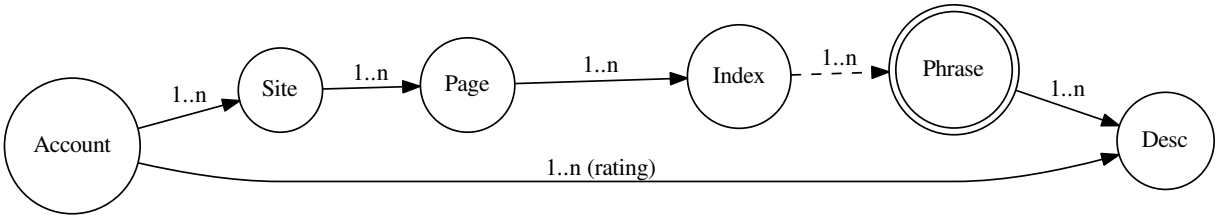

Abbildung 3: Hierarchische Darstellung der zu Grunde liegenden Datenstruktur und Einordnung des Index.

wird. Die Motivation hierfür liegt darin, dass Suchmaschinen diese Datei auch als Basis für die Indizierung der Webseite nutzen [2]. Diese Datei liegt entweder unter `<protocol>://<baseurl>/sitemap.xml` oder ist in der `Robots.txt` angegeben [18] und von dort öffentlich zugreifbar. Der dadurch erstellte Index muss nicht vollständig sein, aber ausreichend für die Erfassung der Inhalte, deren Besuch vom Betreiber gewünscht ist.

(2) Viele Webseiten, die nicht den zuvor genannten Standard nutzen, bieten Besuchern und Suchmaschinen trotzdem eine Möglichkeit an, alle Inhalte des Auftritts auf einer Seite zu sehen. Meistens findet man diese Seiten, indem die Startseite nach dem Ausdruck *Sitemap* durchsucht und der Link verfolgt wird. Durch Extraktion der auf dieser Seite vorkommenden Links lässt sich ebenfalls eine nicht notwendigerweise vollständige Liste aller Einzelseiten der Webseite darstellen, jedoch gilt hier das selbe Prinzip wie bei 1.

(3) Wird keine Art von Sitemap angeboten, so müssen die Seiten manuell durchsucht werden. Dafür wird ein einfaches Prinzip angewandt, das auch Suchmaschinen verwenden: Es werden in der Startseite sämtliche Hyperlinks gesucht und alle externen Links entfernt. Dann werden alle übrigen Seiten geladen und wiederum nach Hyperlinks im geladenen HTML durchsucht. Bereits besuchte Links werden ignoriert. Somit kann in wenigen Iterationen die Seite vollständig indiziert werden. Da hierbei jede Einzelseite aufgerufen, komplett geladen und analysiert wird, ist diese Art der Erfassung die rechenintensivste der genannten Möglichkeiten und sollte nur verwendet werden, wenn keine Art von Sitemap zur Verfügung steht oder der Betreiber die Indizierung explizit wünscht.

6 SPEICHERUNG DER DATEN

Die Struktur der Daten ist insgesamt hierarchisch aufgebaut (vgl. Abb. 3) und hat dabei eine strikte Zuordnung, bei der jeder beteiligte Datensatz eine $1..n$-Zuordnung zu einem anderen besitzt. Da die Daten eine klassische, relationale Struktur aufweisen, wird eine entsprechende Datenbank eingesetzt. Wurde auf Grund der starken Fokussierung auf textuelle Indizes beim Zugriff auf die erfassten Phrasen für den ersten *Proof of Concept (PoC)* auch eine dokumentorientierte Datenbanken verwendet, die auf die Verwendung von Zeichenketten-Indizes hin optimiert sind, ist die relationale Datenhaltung für das Gesamtprojekt sinnvoller. Der Fokus im PoC wurde außerdem auf Grund des Fehlens von Schemata bei Dokumentendatenbanken verwendet, was Erweiterungen ohne strukturelle Anpassungen an der Datenbank möglich macht [6]. Durch die Fähigkeit von modernen relationalen Datenbanken auch schemalose strukturierte Daten zu speichern und indizieren[8, S. 49f], können die Vorteile beider Paradigmen jedoch besser kombiniert werden.

7 ERSTE VERSUCHE

Zu Beginn wurde nur eine sehr kleine Suchbasis angegeben, die aus zufällig ausgewählten Ausdrücken bestanden. Diese waren teilweise in der zu untersuchenden Webseite enthalten und zum Teil nicht. Dadurch sollte sowohl der korrekte Einschluss als auch Ausschluss aus der Ergebnismenge nach Analyse der Webseite bestätigt werden. Die Versuche wurden mit zufällig ausgewählten, tagesaktuellen Seiten von *https://www.tagesschau.de*, *https://www.spiegel.de* und *https://www.heise.de* durchgeführt. Die Texte von den Anbietern *Heise* (*Flesch-Reading-Ease-Score* der Stichprobe[1] von 35) weisen eine stärkere fachliche Prägung auf, wogegen die *Tagesschau* und *Der Spiegel* einen niedrigeren Score aufweisen (28 bzw. 24). In den Versuchen wurde die Dauer des Durchlaufs für eine einzelne Seite gemessen und auf etwa $0,41$ Sekunden gemittelt (20 Versuche). Wurden 10 Seiten

[1] Es wurde eine Stichprobe ($N = 30$) zu einem einheitlichen Thema genutzt.

mit einem Aufruf parallel bearbeitet, so ergab sich bei 10 Messungen eine Dauer von 0, 78s und 30 Seiten parallel wurden bei 5 Versuchen in 2, 18s verarbeitet. Da eine sehr kleine Basis von Ausdrücken verwendet wurde ($|G| = 8$), ist auch die Mächtigkeit des resultierenden Index I niedriger als unter reellen Bedingungen. Die weitere Untersuchung der Benchmarks legt nahe, dass die Analyse den geringsten Anteil an der Gesamtdurchlaufzeit benötigt und der limitierende Faktor eher die Bandbreite der verwendeten Internetverbindung und die Anzahl der verfügbaren Rechenkerne ist. Dies bedeutet, dass eine horizontale Skalierung hier langfristig die Verarbeitungsgeschwindigkeit erhöhen könnte. Für die Verarbeitung von Webseiten privater Anbieter sind diese Zahlen jedoch mehr als ausreichend, da ganze Webauftritte in wenigen Sekunden bis Minuten erfasst werden können.

8 PROBLEME UND OPTIMIERUNGSMÖGLICHKEITEN

Durch die vollständig kontextfreie Verarbeitung der Texte, können Ausdrücke, die aus den gleichen Worten bestehen, vermischt werden. Um diese Problematik langfristig zu vermeiden, wäre der unterstützende Einsatz von Kontext analysierender Software oder linguistischen Algorithmen denkbar [5]. In Kombination mit der Nutzung von Technologien aus dem Bereich *Machine-Learning (ML)*, wäre eine zielgerichtetere Bewertung möglich, als mit rein statischen Untersuchungen, wie sie in dieser Arbeit beschrieben werden [1, 3, 14]. Die Nutzer der Plattform könnten dabei wiederum dazu beitragen, durch das Bewerten von durch den Algorithmus generierten Vorschlägen, die zur Weiterentwicklung benötigten Trainingsdaten zu erfassen. Dadurch wäre der Crowdsourcing-Gedanke direkt auf den Suchalgorithmus anwendbar und somit langfristig in der Lage, die Qualität der Resultate laufend zu steigern. Auf lange Sicht ist daher der Einsatz von linguistischem ML vermutlich die einzige Möglichkeit, kontextabhängige Begriffe mit einer sehr hohen Trefferquote automatisiert zu erfassen.
Weiterhin müssen durch grammatikalische Beugungen (z.B. Konjugation/Deklination) auch von der Stammform abweichende Ausdrücke gefunden werden. Dies erfordert die zusätzliche Erfassung aller möglicher Formen eines Ausdrucks, damit bei der textuellen Suche jede mögliche Form auch gefunden und der Anteil an „vereinfachbaren" Texten maximiert werden kann. Dadurch erhöht sich allerdings die Menge an benötigtem Speicher, sowie der Pflegeaufwand für Ausdrücke um ein Vielfaches. Der Aufwand für die Pflege, wäre durch den Einsatz von freien Wörterbüchern, die eine Verwendung der Daten erlauben (z.B. *Wiktionary Flexikon*) in Grenzen zu halten, indem Vorschläge für Ableitungen von Stammformen automatisiert beim Erstellen eines Ausdrucks angezeigt und vom Bearbeiter einfach übernommen werden können.

9 ZUSAMMENFASSUNG UND AUSBLICK

Mit den hier beschriebenen Methoden lassen sich die Strukturen der Webseiten erfassen und asynchron dazu die Inhalte aller Einzelseiten mit den in der Datenbank vorhandenen Begriffen abgleichen. Das ermöglicht, dass für die Erklärung der auf einer Seite vorhandenen Ausdrücke bei Seitenaufruf nur ein Bruchteil der Daten vom Server geladen werden muss. Das reduziert sowohl die Kosten, die der Service verursacht, als auch die Wartezeit beim Besucher, was für Menschen mit kognitiven Behinderungen notwendig ist, aber auch für alle Menschen einen großen Vorteil mit sich bringt[11, 17, 19].
Für einen optimalen Einsatz der entwickelten Techniken muss nun die korrekte Orchestrierung von Seiten-Struktur- und Seiten-Phrasen-Indizierung stattfinden. Dafür muss der optimale Mittelweg zwischen Aktualität des Index und Kosten des Indizierungsprozesses (z.B. durch Prozessorlast und Traffic) gefunden werden. Eine Möglichkeit wäre die Nutzung der im *sitemap.org*-Standard vorgeschlagenen Aktualisierungshäufigkeit. Damit hat der Seitenbetreiber seine Aktualität selbst in der Hand. Auch eine Deckelung der Häufigkeit auf kosteneffiziente Zeiten wie z.B. eine Woche mit der Option einer Monetarisierung häufigerer Durchführung wäre ein mögliches Szenario.
Weiterhin muss sich zeigen, ob es ausreicht das Durchsuchens nach Ausdrücken auf dem gleichen Server mit einer durchschnittlichen Anzahl von 4-8 Rechenkernen auszuführen, oder ob dedizierte Suchserver mit einer hohen Anzahl an Kernen benötigt wird. Dafür muss jedoch die Anzahl an Teilnehmern und die Menge an erfassten Ausdrücken zunächst noch ansteigen. Für eine derartige vertikale Skalierung kämen zwei mögliche Szenarien in Frage: Einerseits die Portierung des Suchalgorithmus auf GPU (Graphics Processing Unit)-Chips durch Technologien wie z.B. *CUDA* [4]. Da die Berechnungen jedoch primär für 16-Bit Fließkommaoperationen optimiert sind, ist es fraglich, ob die Steigerung der Performanz den höheren Stromverbrauch aufwiegt. Die zweite Möglichkeit wäre eine Portierung für einen Einsatz in der Cloud. Der große Vorteil hierbei ist, dass Anbieter wie *Google Cloud Computing* und *Amazon Web Services* eine Möglichkeit der On-Demand-Provisionierung von virtuellen Servern mit einer beliebigen Anzahl von Kernen anbieten. Besonders für Anbieter mit großen Datenmengen wäre diese Option in Kombination mit der Monetarisierung ein Kosteneffizienter Weg.
Die Implementierung der Suche erfolgte bereits parallel zur Erstellung dieses Beitrags in C++ und wird 2019 im Detail getestet und bei Bedarf erweitert werden, da es sich um einen zentralen Teil im Projekt *Ordiphrase* handelt.

LITERATURVERZEICHNIS

[1] Asad Abdi, Siti Mariyam Shamsuddin, Shafaatunnur Hasan, and Jalil Piran. 2018. Machine learning-based multi-documents sentiment-oriented summarization using linguistic treatment. *Expert Systems with Applications* 109 (Nov. 2018), 66–85. https://doi.org/10.1016/j.eswa.2018.05.010

[2] Jo Bager. 2006. Gemeinsamer Sitemaps-Standard von Google, Microsoft und Yahoo. https://www.heise.de/-118488.html

[3] Philipp Cimiano, Günter Ladwig, and Steffen Staab. 2005. Gimme' the Context: Context-driven Automatic Semantic Annotation with C-PANKOW. In *Proceedings of the 14th International Conference on World Wide Web (WWW '05)*. ACM, New York, NY, USA, 332–341. https://doi.org/10.1145/1060745.1060796 event-place: Chiba, Japan.

[4] Cyril Zeller. 2011. CUDA C/C++ Basics. http://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf

[5] M Diligenti, F M Coetzee, S Lawrence, C L Giles, and M Gori. 2000. Focused Crawling Using Context Graphs. *VLDB* 2000 (Sept. 2000), 527–534.

[6] Cornelia Gyorodi, Robert Gyorodi, George Pecherle, and Andrada Olah. 2015. A comparative study: MongoDB vs. MySQL. In *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*. IEEE, Oradea, Romania, 1–6. https://doi.org/10.1109/EMES.2015.7158433

[7] Mathias Haimerl. 2019. Einfach Digitalisierung – Konzept einer universellen Simplifizierung des digitalen Lebens. *FifF-Kommunikation* 4/18 (Jan. 2019), 34–38.

[8] Salahaldin Juba and Andrey Volkov. 2019. *Learning PostgreSQL 11: a beginner's guide to building high-performance PostgreSQL database solutions* (3 ed.). OCLC: 1085152308.

[9] Gudrun Kellermann. 2014. Leichte und Einfache Sprache – Versuch einer Definition | bpb. http://www.bpb.de/apuz/179341/leichte-und-einfache-sprache-versuch-einer-definition

[10] Donald Ervin Knuth. 1997. *The art of computer programming – Volume 2: Sorting and Searching* (2 ed.). The art of computer programming, Vol. 3. Addison-Wesley, Reading, Mass.

[11] Steve Krug. 2014. *Don't make me think! Web & mobile usability - das intuitive Web* (dritte auflage ed.). mitp, Frechen. OCLC: 894719048.

[12] John Levine. 2009. *flex & bison* (1 ed.). O'Reilly and Associates, Newton, MA.

[13] Jiří Maršík and Ondřej Bojar. 2012. TrTok: A Fast and Trainable Tokenizer for Natural Languages. *The Prague Bulletin of Mathematical Linguistics* 98, 1 (Oct. 2012), 75–85. https://doi.org/10.2478/v10108-012-0010-0

[14] Kemal Ofazer, Sergei Nirenburg, and Marjorie McShane. 2001. Bootstrapping Morphological Analyzers by Combining Human Elicitation and Machine Learning. *Computational Linguistics* 27, 1 (2001), 59–85. https://doi.org/10.1162/089120101300346804

[15] Gautam Pant and Padmini Srinivasan. 2005. Learning to crawl: Comparing classification schemes. *ACM Transactions on Information Systems* 23, 4 (Oct. 2005), 430–462. https://doi.org/10.1145/1095872.1095875

[16] Gautam Pant, Padmini Srinivasan, and Filippo Menczer. 2004. Crawling the Web. Springer Berlin Heidelberg, Berlin, Heidelberg, 153–177. https://doi.org/10.1007/978-3-662-10874-1_7

[17] Peter Rasche, Alexander Mertens, Christopher Schlick, and Pilsung Choe. 2015. Taktiles Feedback zur alternsgerechten Gestaltung der Interaktion mit mobiler Informations- und Kommunikationstechnologie.

[18] sitemaps.org. 2016. sitemaps.org - Protocol. https://www.sitemaps.org/protocol.html

[19] Eberhard Zehendner. 2019. Wenn's mal wieder etwas länger dauert Session Timeouts als Barriere für ältere Menschen. *FifF-Kommunikation* 4/18 (Jan. 2019), 50–56.