technische universität
dortmund

PG-588: Endbericht

SciencePlag

15. Mai 2016

Teilnehmer:
Ole Bergenholtz
Jonas Holtorf
Marc Jasper
Raoul Lefmann
Kevin Nikiel
Lutz Oettershagen
Philip Rehm

Betreuer:
Prof. Dr. Johannes Fischer
Prof. Dr. Sven Rahmann
Dominik Köppl
Florian Kurpicz

Technische Universität Dortmund - Fakultät für Informatik
Lehrstuhl XI - Algorithm Engineering - http://ls11-www.cs.tu-dortmund.de

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Spätestens nach dem Beginn der guttenbergschen Plagiatsaffäre im Jahr 2011 ist ein verstärktes öffentliches Interesse an der Aufdeckung von geistigem Diebstahl in Form von Plagiarismus wahrzunehmen. Plagiarismus bezeichnet hier den textuellen Diebstahl geistigen Eigentums, bei dem der Wortlaut von einer anderen Person mehr oder weniger unverändert übernommen wird, ohne diese als Urheber zu nennen. Das Finden der Plagiate in der Doktorarbeit von Guttenberg wurde zur gemeinsamen Aufgabe von Freiwilligen, welche sich über das *GuttenPlag-Wiki*[1] vereinten. Diese Teamarbeit wurde zum Vorbild weiterer Zusammenschlüsse, wie z.B. der kollaborativen Plattform *VroniPlag*[2], auf welcher bis zum Zeitpunkt der Fertigstellung dieses Berichts schon über 150 Dissertationen und andere wissenschaftliche Werke auf Plagiarismus untersucht wurden. In all diesen Arbeiten wurden auf einer zweistelligen Prozentzahl der Seiten Plagiate gefunden.

Das Finden von Plagiaten ist ein für Menschen sehr langwieriges Unterfangen, welches fachspezifische Experten und Personen mit großem Wissen über bereits bestehende Literatur und deren Wortlaut benötigt. Eine logische Konsequenz ist, dass die Aufgabe zum Teil durch Computerprogramme automatisiert werden sollte, können diese doch riesige Mengen an Daten in kürzester Zeit verarbeiten. Es besteht also ein Bedarf an unterstützender Software für die Plagiatssuche. Die Anwendung solcher Programme ist nicht nur auf Zusammenschlüsse wie VroniPlag etc. begrenzt, sondern für eine Vielzahl von Einrichtungen relevant, wie z.B. Schulen, Universitäten und allgemein in der Forschung. Nur wenn es möglich ist, mit solchen automatischen Programmen mit hoher Wahrscheinlichkeit Plagiate zu erkennen, wird der akademische Plagiarismus auf lange Sicht reduziert werden können.

1.1 Zielsetzung der Projektgruppe

Ziel der Projektgruppe *SciencePlag* ist die Konzeption und Umsetzung einer Software zur effizienten Plagiatserkennung [5]. Methodisch soll dies durch eine Kombination von Konzepten aus *Stringology* und *Information Retrieval* geschehen. Die Stringology ist das Gebiet, dass sich mit der Analyse von Zeichenketten und dem Entwurf von String-Algorithmen beschäftigt. Techniken des Information Retrieval ermöglichen die automatische Extraktion von strukturiertem Wissen aus unstrukturierten meist natürlichsprachi-

[1] `http://de.guttenplag.wikia.com/wiki/GuttenPlag_Wiki` (Zuletzt abgerufen: 06.04.2016)
[2] `http://de.vroniplag.wikia.com/wiki/Home` (Zuletzt abgerufen: 06.04.2016)

gen Dokumenten. Es dient zusammen mit einer Vorverarbeitung der Dokumente dazu, eine Vorauswahl möglicher Plagiatsquellen zu treffen.

Aus dem Bereich der Stringology soll eine Indexstruktur gewählt werden, um ein effizientes Text Alignment, das heißt eine effiziente Suche nach möglichen Plagiaten in den vom Information Retrieval gefundenen Dokumenten, zu realisieren. Um Plagiate finden zu können, benötigt die Software offensichtlich Zugriff auf die Dokumente, aus denen plagiiert wurde. Daher muss ihr eine ausreichend umfassende Datenbasis mit Dokumenten zur Verfügung stehen, aus welchen Plagiate entstammen könnten. Um eine eine effiziente Suche nach Dokumenten, aus denen potenziell plagiiert worden sein kann, zu ermöglichen, werden die Dokumente der Datenbasis vorverarbeitet.

Unsere Software SciencePlag setzt ein mehrstufiges Vorgehen um. Zunächst wird eine statische Datenbasis erstellt, welche z. B. eine Menge domänenspezifischer Arbeiten enthält. Diese Datenbasis wird im Anschluss für die Suche nach Dokumenten, aus denen Plagiate in einem verdächtigen Dokument stammen könnten, genutzt. Die gefundenen Dokumente werden daraufhin in einer Indexstruktur abgelegt, welche letztendliche Suche nach Plagiaten effizient ermöglicht.

1.2 Aufbau des Berichts

Abschnitt 2 stellt zunächst die für das Verständnis dieses Berichts benötigten Grundlagen vor. Insbesondere werden Methoden des Information Retrievals und die Datenstruktur Suffixarray betrachtet, welche wesentliche Elemente unseres Lösungsansatzes sind. In Kapitel 3 werden bereits bestehenden Lösungsansätze für die Detektion von Plagiaten vorgestellt. Hierbei wird neben den Gewinnern des PAN-Wettbewerbs auch auf kommerziell angebotene Softwareprodukte kurz eingegangen. Die von unserer Projektgruppe gewählte Vorgehensweise ist Thema von Kapitel 4. Auch werden die von uns entwickelten Algorithmen im Detail erläutert. Eine Anleitung zur Verwendung der Software erhält der Leser in Kapitel 5. Dort wird auch ein Skript vorgestellt, welches dabei hilft, die Artikel der Online-Enzyklopädie Wikipedia als Quelldokumente in der Datenbasis zu verwenden. Eine umfassende Evaluation unserer Software folgt in Abschnitt 6 vorgenommen. Die abschließende Betrachtung der Ergebnisse, sowie des Ablaufs der Projektgruppe und ein Fazit mit Ideen, wie SciencePlag in Zukunft erweitert werden kann, schließen den Bericht ab.

Im Anhang befinden sich zusätzlich eine vollständige Beschreibung der verwendeten Entwicklungsumgebung, sowie eine detaillierte Installationsanweisung der SciencePlag-Anwendung (Anhänge A bzw. B). Dort befindet sich auch eine Auflistung der von uns verwendeten Stoppwörter (Anhang C).

KAPITEL 2

Grundlagen

In diesem Kapitel werden wir zunächst erklären, was wir unter einem Plagiat verstehen und welche verschiedenen Arten von Plagiaten es gibt. Wir beschreiben das typische Vorgehen bei der automatisierten Plagiatsfindung und gehen dabei auch auf ihre Grenzen ein. Anschließend liefern wir einen Überblick über benötigte Aspekte des Information Retrieval, ehe einige Grundlagen zu Indexstrukturen vorgestellt werden, die wichtig für die effiziente Durchführung der Plagiatserkennung sind. Im Rahmen dieses Kapitels werden Begriffe definiert, die zum Verständnis der übrigen Abschnitte des Berichtes notwendig sind.

2.1 Plagiate

Plagiate bezeichnen die Übernahme fremden geistigen Eigentums, ohne die Quelle zu kennzeichnen. Dabei kann zwischen verschiedenen Arten des Plagiats unterschieden werden [23]. Wir beschränken uns im Folgenden auf textbasierte Plagiate, allerdings sind z. B. auch Plagiate von Bildern möglich. Die einfachste Form des Plagiats ist das *Copy-&-Paste*-Plagiat. Dabei werden Passagen aus dem Originaltext ohne Veränderungen übernommen. In extremen Fällen werden dabei sogar Rechtschreibfehler des Originals kopiert. Copy-&-Paste-Plagiate lassen sich in der Regel leicht erkennen. Eine Weiterentwicklung des Copy & Paste ist das *Verschleierungsplagiat*. Dabei werden einzelne Wörter durch Synonyme ausgetauscht oder die Reihenfolge einzelner Sätzen umgestellt. Plagiate dieses Typs sind schwieriger zu finden. Eine Kombination und Vermischung verschiedener Originalpassagen zu einem einzigen Text nennt sich *Shake & Paste*. Plagiate solcher Art fallen oft durch einen wechselnden Schreibstil auf.

Eine *Übersetzung* einer Textstelle aus dem Original einer anderen Sprache ohne Nennung der Quelle ist ebenso eine Form des Plagiats. Dabei hofft der Autor des Plagiats darauf, dass die Quelle obskur genug ist, damit das Plagiat nicht auffällt. Plagiate dieser Form sind sehr schwer mit Hilfe von Software zu finden, da derzeitige Programme zur Übersetzung von Texten noch nicht ausgereift sind, und Übersetzungen von unterschiedlicher Software meist auch in unterschiedlichen Texten resultieren.

Weitere Formen des Plagiats sind *strukturelle Plagiate*, bei denen der Originaltext paraphrasiert oder die Argumentationsstruktur des Originals übernommen wird. Bei dieser Form rechtfertigt sich der Plagiator, indem er argumentiert, dass dies kein Plagiat sei, da die Quelle nicht wörtlich wiedergegeben wird. Da aber auch bei einem Plagiat dieser

Form geistiges Eigentum eines anderen ohne Attribution verwendet wird, handelt es sich ebenfalls um eine Plagiatsvariante. Das *Bauernopfer* ist eine Form des Plagiats, bei der ein Teil des Originals korrekt zitiert wird, aber weitere Teile ohne Kennzeichnung übernommen werden. Bei einem *Eigenplagiat* werden Teile von früheren Texten des Autors ohne Kennzeichnung übernommen.

In Deutschland werden alle sprachlichen Werke, unter anderem auch wissenschaftliche Publikationen, durch das Urheberrechtsgesetz (UrhG) geschützt. Bei der Verwendung eines veröffentlichten Werks, z.B. als Zitat (§ 51 UrhG), ist „die Quelle deutlich anzugeben“ (§ 63 UrhG). Geschieht dies nicht, wird eine Urheberrechtsverletzung (§ 106 UrhG) begangen, welche mit bis zu drei Jahren Gefängnis oder einer Geldstrafe bestraft wird. Darüber hinaus hat der Urheber des Werks unter anderem Ansprüche auf Unterlassung, Schadenersatz, Rückruf und Entschädigung (§§ 97–105 UrhG).

Universitäten verlangen bei der Abgabe einer Bachelor-, Master- oder Doktorthesis zusätzlich eine eidesstattliche Versicherung des Autors, um Plagiaten vorzubeugen. Im Strafgesetzbuch ist nämlich die Abgabe einer falschen Versicherung mit bis zu drei Jahren Freiheitsstrafe oder einer Geldstrafe bewehrt (§ 156 StGB). Dies bedeutet, dass ein Plagiat neben den zivilrechtlichen auch strafrechtliche Konsequenzen haben kann. Darüber hinaus verstoßen Plagiate für gewöhnlich auch gegen die Richtlinien der Universität und werden mit dem Entzug des erworbenen akademischen Grades oder dem Herabsetzen der Note geahndet.

Plagiate werden nicht nur bewusst aus fehlendem Respekt gegenüber der wissenschaftlichen Praxis angefertigt. Sie entstehen auch, wenn beim Autor kein ausreichendes Bewusstsein dafür existiert, dass es sich dabei um akademisches Fehlverhalten handelt, oder wenn unter Zeit- oder Leistungsdruck die korrekte Arbeitsweise leidet.

2.2 Plagiatsfindung

Aufgabe der Plagiatsfindung ist es, in einem Dokument mögliche Plagiatsstellen zu finden. Ein solches Dokument, welches im Verdacht steht Plagiate zu enthalten, wird im Folgenden *verdächtiges Dokument* genannt. Dafür steht eine große Menge von Dokumenten zur Verfügung, mit denen wir das verdächtige Dokument vergleichen können. Diese bezeichnen wir als *Dokumentenbasis* und ihre Elemente als *Quelldokumente*. Man unterscheidet zwischen statischen und dynamischen Dokumentenbasen. Eine *statische Dokumentenbasis* ist eine Datenbank von Dokumenten, in der alle Dokumente, die zu einem Zeitpunkt zur Plagiatsfindung verwendet werden, bereits vorher bekannt sind. Eine statische Dokumentenbasis bedeutet also nicht, dass sie nicht durch neue Dokumente erweitert werden kann, sondern das dies getrennt von der Plagiatsfindung stattfinden muss. Eine *dynamische Dokumentenbasis* hingegen erlaubt das Finden zusätzlicher Dokumente während der Plagiatsfindung, z.B. durch eine Internetsuche.

Eine vollständig automatisierte Plagiatsfindung durch eine Computersoftware ist sehr schwierig. Zum einen müssten einem solchen Programm dafür alle Dokumente zur Verfügung stehen, aus denen überhaupt plagiiert werden kann. Viele dieser Dokumente liegen nicht in digitalisierter, also maschinenlesbarer, Form vor. Man stelle sich vor, eine Person A schreibt heimlich einen Absatz aus dem handschriftlichen Tagebuch einer anderen Person B ab, welches ansonsten in einer Schreibtischschublade in der Wohnung von B sicher verstaut ist. Um dieses Plagiat zu erkennen, müsste ein Erkennungsprogramm Kenntnis über die Inhalte des Tagebuchs erhalten können. Weiterhin lässt sich auch nicht nur aus Dokumenten plagiieren, sondern z.B. auch wortwörtlich Passagen aus persönlichen Korrespondenzen. Die Software müsste somit neben jedem Dokument auch noch Zugriff auf

jedes Gespräch, was jemals zwischen Menschen geführt wurde, haben. Eine perfekte Plagiatsfindung würde insofern Zugriff auf sämtliches Wissen unserer Welt benötigen, in der Realität steht ihr jedoch nur eine endliche Teilmenge davon zur Verfügung. Ein Programm für die Plagiatsfindung wird also nicht alle Textstellen ausgeben können, die tatsächlich Plagiate sind. Auf der anderen Seite werden in manchen Fällen Textstellen ausgegeben, bei denen es sich nicht um Plagiate handelt. Das hat zum einen damit zu tun, dass man viele Textstellen nicht eindeutig als plagiiert oder nicht-plagiiert klassifizieren kann. Es existiert eine Grauzone, innerhalb derer es Auslegungssache eines Prüfers ist, ob es sich um ein Plagiat handelt. Das Programm selbst kann zur Zeit noch nicht der Prüfer sein, da die Akzeptanz von Verfahren zur Plagiatsfindung noch nicht groß genug ist. Beispielsweise würde eine Universität einen Doktorgrad nicht ungeprüft einzig aufgrund der Tatsache entziehen, dass eine Maschine Plagiate in einer Doktorarbeit gefunden hat. Zudem gibt es noch zahlreiche weitere Gründe für derartige Fehler. Ein Beispiel sind sogenannte *häufige Formulierungen*, also Sätze oder Bestandteile von Sätzen, die in zahlreichen Dokumenten in gleicher oder ähnlicher Form vorkommen. Diese können nur dann erkannt werden, wenn sie in der Dokumentenbasis häufig vorkommen, was nicht für jede solche Formulierung garantiert werden kann, denn dafür müsste die Dokumentenbasis eine ausreichend große, repräsentative Teilmenge aller Dokumente sein. Auch kann es passieren, dass der Verfasser des verdächtigen Dokuments durch Zufall ohne sein Wissen einzelne Formulierungen ähnlich gewählt hat, wie der Autor eines Quelldokumentes. Mit steigender Größe der Dokumentenbasis steigt die Wahrscheinlichkeit für solche Zufallstreffer, was als eine Ausprägung von Bonferronis Prinzip angesehen werden kann (siehe [18] Abschnitt 1.2.2). Dieses besagt, dass wenn die Häufigkeit eines Ereignisses (hier: Plagiat) in einem zufällig gezogenen Datensatz (hier: zufällig generierte Dokumente), groß genug ist, dann ist zu erwarten, dass ein signifikanter Anteil der vorhergesagten Auftreten eines Ereignisses False-Positives sind.

Aus diesem Grund liefern Plagiatserkennungsprogramme zur Zeit lediglich eine Vorauswahl von Textstellen, die möglicherweise Plagiate enthalten. Das Resultat der Plagiatsfindung nennen wir daher *verdächtige Stellen* und nicht Plagiatsstellen. Sie bestehen aus Verweisen auf eine Stelle im verdächtigen Dokument, sowie auf eine zugehörige Stelle aus einem Dokument, aus dem möglicherweise plagiiert wurde. Die letztliche Entscheidung, ob es sich bei jeder einzelnen verdächtigen Stelle tatsächlich um ein Plagiat handelt, liegt immer beim Nutzer des Plagiatserkennungsprogramms, der die beiden von der Software ausgegebenen Textstellen in den jeweiligen Dokumenten miteinander vergleicht. Es wird im Folgenden nicht mehr strikt zwischen verdächtigen Stellen, tatsächlichen Plagiatsstellen oder *Fundstellen* unterschieden, es sei denn, der Unterschied soll explizit hervorgehoben werden.

Oftmals wird das Finden von Plagiaten in zwei unabhängige Teilschritte aufgeteilt: das Information Retrieval und die eigentliche Plagiatsfindung. Dieses Vorgehen hat sich in bestehender Plagiatsfindungssoftware (wie den in Abschnitt 3 vorgestellten Programmen) bewährt. Beim *Information Retrieval* wird eine im Verhältnis zur Gesamtzahl an Quelldokumenten kleine Anzahl von sogenannten *Kandidatendokumenten* aus der Dokumentenbasis ausgewählt, die dem verdächtigen Dokument bezüglich eines schnell berechenbaren Ähnlichkeitsmaßes am ähnlichsten sind. Der Grund dafür ist, dass es nicht praktikabel wäre, alle Dokumente einer potenziell sehr großen Dokumentenbasis stellenweise mit dem verdächtigen Dokument zu vergleichen. Da in diesem Schritt Kandidaten für die eigentliche Plagiatssuche ermittelt werden, bezeichnen wir ihn auch als *Candidate Retrieval*.

Bei der eigentlichen Plagiatsfindung werden die im Candidate Retrieval gefundenen Kandidatendokumente dann im Detail mit dem verdächtigen Dokument verglichen, um Plagiate zu finden. Ein einfacher String-Matching-Algorithmus genügt hier nicht, denn

Autoren plagiierter Schriftstücke neigen dazu, Texte nicht wortwörtlich zu kopieren, sondern die Wortstellung zu verändern, Synonyme zu verwenden oder die plagiierten Stellen anderweitig zu verschleiern (vgl. mit den in Abschnitt 2.1 vorgestellten Ate von Plagiaten). Da in diesem Schritt Stellen in verdächtigem und Kandidatendokument miteinander abgeglichen werden, nennen wir diesen Schritt *Text Alignment*. Zur Beschleunigung des Text Alignment sind Indexstrukturen, wie z.B. Suffixarrays, nützlich (siehe Abschnitt 2.5).

2.3 Vorverarbeitung

Den meisten Verfahren zur Verarbeitung von natürlichsprachigen Texten wird ein sogenannter *Vorverarbeitungsprozess* vorangestellt, der auch oft als *Preprocessing* bezeichnet wird. Sein Ziel ist die Verbesserung der Qualität der Resultate in den nachfolgenden Schritten. Bei der Plagiatsfindung ist eine Vorverarbeitung der Quelldokumente und des verdächtigen Dokuments notwendig um hohe Erkennungsraten zu erzielen und weniger anfällig für einfache Verschleierungen zu sein. Eine typische Vorverarbeitung von Texten setzt sich aus den Schritten *Segmentierung*, *Stammformreduktion* und *Stoppwörterentfernung* zusammen. Zunächst wird der Text aller für das Verfahren relevanter Dokumente in einzelne Worte aufgeteilt. Nun werden diese jeweils auf ihren Wortstamm reduziert und anschließend nicht benötigte Füllwörter herausgefiltert. Dieser Abschnitt soll zunächst allgemeine Grundlagen des Preprocessing behandeln, Details unseres Ansatzes werden in Abschnitt 4.3 erläutert.

Die Textvorverarbeitung erwartet eine Zeichenkette über einem *endlichen Alphabet* Σ als Eingabe [8, 3]. Im Deutschen enthält Σ typischerweise mindestens die Buchstaben A–Z und die Sonderzeichen Ä, Ö, Ü und ß, sowie die entsprechenden Kleinbuchstaben. Für die englische Sprache sind hingegen weniger Zeichen notwendig. Im Allgemeinen werden aber mehr Zeichen für die Repräsentation genutzt. Diese können z.B. Satzzeichen und Sonderzeichen umfassen. Zeichen werden in einem Text durch Bytefolgen kodiert. Bei der klassichen ASCII-Kodierung reicht für jedes kodiertes Zeichen ein Byte aus. Während für die englische Sprache oft der ASCII-Zeichensatz genügt, muss für die korrekte Wiedergabe von deutschen Texten eine allgemeinere Kodierung verwendet werden. Unser Programm erwartet Eingaben, die nach dem UTF-8-Standard kodiert sind. UTF-8 kodiert typische westliche Zeichen mit ein oder zwei Bytes und bietet die Möglichkeit, auch alle anderen vom Unicode-Konsortium[1] definierten Zeichen zu kodieren. Eine *Eingabezeichenkette* lässt sich dann über dem Eingabealphabet Σ als $t \in \Sigma^*$ darstellen.

2.3.1 Segmentierung von Text

Die Segmentierung von Texten hat zum Ziel diese in sinnvolle Einheiten zu zerlegen [8, 3]. Das können Buchstaben, Wortformen, Sätze, Absätze, Abschnitte oder ganze Kapitel sein. Die Segmentierung erhält als Eingabe einen Text, bestehend aus einer Zeichenkette t über einem endlichen Alphabet Σ.

Am Häufigsten wird ein Text in einzelne Wörter aufgespalten, die auch als *Token* bezeichnet werden. Entsprechend wird dieser Prozess *Tokenisierung* genannt. Das Problem der Zerlegung eines Textes in Wortformen scheint zunächst einfach lösbar zu sein, indem der Text an Leerzeichen und Satzzeichen gespalten wird und die entstehenden Teiltexte als Wortformen aufgefasst werden. Sowohl im Deutschen, als auch im Englischen lassen sich allerdings viele Beispiele finden, die zeigen, dass diese Methode oft suboptimal ist, da

[1] `http://unicode.org/` (Zuletzt abgerufen: 06.04.2016)

sie nicht immer zu sinnvollen Wortformen führt. Beispielsweise würde die Abkürzung *z.B.* vom oben beschriebenen naiven Tokenisierungsprozess in zwei Token z und B zerlegt.

Idealerweise werden bei der Tokenisierung Namen von Entitäten nicht aufgespalten. Eine *Entität* ist etwas, dem man einen Namen geben kann, z.B. eine bestimmte Person, eine bestimmte oder abstrakte Sache, oder ein Ereignis. In Textdokumenten werden Entitäten durch *Entitätsnamen* referenziert, die nicht eindeutig sein müssen und aus einem oder mehreren Wörtern bestehen können. So bezeichnen z.B. die Zeichenketten *International Business Machines* und *IBM* die gleiche Entität. Schon an diesem Beispiel ist zu erkennen, dass es suboptimal sein kann, die erste der beiden Zeichenketten in drei Token aufzuteilen, da dadurch jegliche Möglichkeit verlorengeht festzustellen, dass es sich bei den Wortfolgen um Synonyme handelt. Desweiteren kann eine naive Zerlegung des Textes zu Token führen, bei denen es sich nicht um sinnvolle natürlichsprachige Wörter handelt. Die Segmentierung der Zeichenkette *Vor- und Nachteile* z.B. würde das Token *Vor-*, anstelle des korrekten *Vorteil* liefern. Verfahren zur bestmöglichen Tokenisierung erfordern sprachabhängige Informationen wie z.B. Wörterbücher oder nutzen Methoden der Informationsextraktion wie z. B. Named-Entity-Recognition-Verfahren (eine Übersicht ist in [21] zu finden).

2.3.2 Stammformreduktion

Die *Stammformreduktion* ist ein wichtiger Schritt in der weiteren Vorverarbeitung der gewonnenen Wortformen, bei dem diese auf ihren *Wortstamm* reduziert werden [8, 3]. Viele Zeichenketten aus natürlichsprachigen Texten tragen ähnliche Information, so z.B. die Wörter *Heizung*, *heizen*, *Heizer* und *heizten*. Alle diese Wörter haben den selben Wortstamm. Sie auf diesen zu reduzieren ist das Ziel der Stammformreduktion. Der Stamm eines Wortes ist häufig, aber nicht immer eines seiner Präfixe. Es ist wichtig anzumerken, dass in den jeweiligen Endungen der Wörter durchaus Information steckt. Die Wörter *Heizung* und *heizen* tragen nicht exakt die selbe Information, so ist eines davon ein Substantiv, das andere ein Verb. Dennoch wird angenommen, dass grammatikalische Endungen eines Wortes zu vernachlässigen sind. Diese Vereinheitlichung hat sich vor allem beim Finden von Verschleierungsplagiaten als nützlich erwiesen. Ein Stammformreduktionsprozess könnte beispielsweise die oben genannten Worte alle auf den Stamm *heiz* reduzieren.

Bei der Stammformreduktion, die auch als *Stemming* bezeichnet wird, können grundsätzlich zwei Arten von Problemen auftreten: *Überreduktion* und *Unterreduktion*. Während bei der Überreduktion zu viel der Wortendung entfernt wird und Wörter mit eigentlich verschiedenem Stamm auf den selben Stamm reduziert werden, wird bei der Unterreduktion zu wenig der Wortendung normalisiert, sodass zwei Wörter mit eigentlich gleichem Stamm auf verschiedene Stämme reduziert werden. Ein Fall von Überreduktion ist beispielsweise, wenn die Token *Wand* und *wandere* auf *wand* reduziert werden. Überreduktion geht mit dem Verlust wichtiger Information einher. Unterreduktion tritt z.B. dann auf, wenn *absorbieren* auf *absorb* und *Absorption* auf *absorp* reduziert werden. Trotz hoher semantischer Ähnlichkeit werden unterschiedliche Stämme gefunden.

Bei dem Textvorverarbeitungsschritt der Stammformreduktion besteht eine hohe Abhängigkeit zu der Sprache, in welcher der Text vorliegt. So werden Wortstämme im Deutschen anders gebildet als im Englischen. Algorithmen für die Stammformreduktion werden als *Stemmer* bezeichnet [2]. Verwendete Versionen sind z. B. der Lancaster-Stemmer [14], der Porter-Stemmer [15] und die verschiedenen Varianten des Snowball-Stemmers [16].

2.3.3 Stoppwörterentfernung

Neben Wörtern, die eine hohe Ähnlichkeit aufweisen, gibt es Wörter, die grundsätzlich geringen spezifischen Informationsgehalt haben. Diese Wörter werden *Stoppwörter* genannt [8, 3]. Durch die Entfernung dieser Stoppwörter können oft bessere Resultate erzielt werden. Insbesondere wird dadurch vermieden, dass diesen Wörten in folgenden Verarbeitungsschritten zu hohe Bedeutung beigemessen wird. Beispiele für Stoppwörter im Deutschen sind *ist*, *aber*, *in* und *an*. Im Englischen lassen sich vergleichbare Stoppwörter wie *is*, *but*, *in* und *at* finden. Eine vollständige Liste der von unserer Software entfernten deutschen und englischen Stoppwörter kann in Anhang C eingesehen werden.

2.4 Information Retrieval

Beim *Information Retrieval* (IR) wird das Ziel verfolgt, automatisch Wissen aus einer Datenbasis zu extrahieren. Es handelt sich um einen weit gefassten Begriff, unter anderem werden folgende Anwendungsgebiete dem IR zugeordnet [6]:

- *Adhoc-Suche*: Ein Benutzer sucht spontan anhand von Suchbegriffen nach relevanten Informationen, z.B. nach Websites.
- *Klassifikation/Clustering*: Ein Dokument soll automatisch in ein vorgegebenes Schema eingeordnet werden. Als einfaches Beispiel dient hier die Einordnung in Ham- bzw. Spam-Mails.
- *Informationsextraktion*: Bestimmte Daten oder Zusammenhänge sollen einem Dokument automatisch entnommen werden.
- *Frage-Antwort-Systeme*: Auf eine (formale) Frage soll eine passende Antwort bereitgestellt werden. Es wird hier nicht nach Quellen für eine Lösung gesucht, sondern direkt nach einer Antwort.

Abbildung 2.1 stellt den Ablauf eines IR-Prozesses schematisch dar. Wichtiger Bestandteil des Systems ist die Repräsentation von Dokumenten aus der Datenbasis D_1 und die der Anfragen Q.

Ist die Repräsentation von Dokumenten und Anfrage definiert, so wird noch ein Ähnlichkeitsmaß benötigt. Basierend auf diesem Maß soll häufig ein Ranking der Dokumente aus D_1 vorgenommen und die zur Anfrage ähnlichsten Einträge ausgegeben werden. Das Ergebnis des IR-Prozesses ist demnach die Teilmenge D_2 der Datenbasis, welche die zur Anfrage ähnlichsten Dokumente enthält.

Legt man einem IR-Ansatz das *Vektorraummodell* zugrunde, so werden sowohl Dokumente aus der Datenbasis als auch die Anfrage als Vektoren repräsentiert. Bei der Verarbeitung von Textdokumenten besteht ein solcher Vektor aus gewichteten Worthäufigkeiten (*Termfrequenzen*) des jeweiligen Dokumentes. Angenommen wir betrachten einen Wortschatz, der aus lediglich zwei Wörtern (Termen) besteht. Dann können sämtliche Dokumente zunächst als Paare (t_1, t_2) der noch ungewichteten Worthäufigkeiten t_1 und t_2 darggestellt werden. Abbildung 2.2 illustriert diese Darstellung anhand der Dokumentenvektoren $d_1 = (1, 2)$ und $d_2 = (3, 1)$. Die Gewichtung der Termfrequenz-Vektoren erfolgt durch Division aller Einträge durch das größte Element im jeweiligen Vektor. Die entsprechenden gewichteten Termfrequenzvektoren sind daher $d_1^w = (\frac{1}{2}, 1)$ und $d_2^w = (1, \frac{1}{3})$.

Für die Projektgruppe verwenden wir das weit verbreitete *tf-idf-Modell* (term frequency - inverse document frequency [19]) zur Gewichtung der Termfrequenzen. Elemente eines Dokumentenvektors sind dabei einzelne tf-idf-Gewichte, die wie folgt berechnet werden.

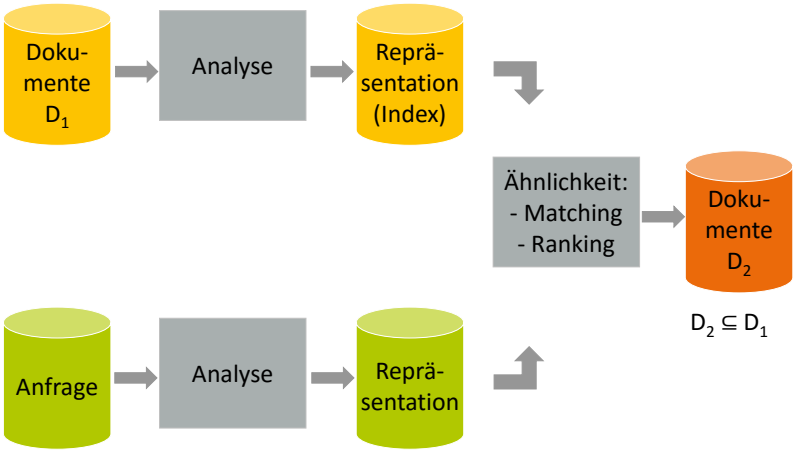


Abbildung 2.1: Information Retrieval als Flow Chart [7]

2.1 Definition (tf-idf-Gewichtung). Sei D die Menge bekannter Dokumente. Sei $q = ((t_1, f_1), (t_2, f_2), ..., (t_n, f_n))$ ein Vektor aus Paaren der Form (Term, Termfrequenz). Sei (t, f) ein beliebiges Paar aus q. Ferner sei $D_t \subseteq D$ die Menge von Dokumenten, in denen der Term t vorkommt. Sei $f_{\max}$ die höchste Termfrequenz im Vektor q. Dann berechnet sich das tf-idf-Gewicht von t in q wie folgt:

$$\text{tf-idf}\left(f, f_{\max}, |D_t|, |D|\right) = \frac{f}{f_{\max}} \cdot \log\left(\frac{|D|}{|D_t|}\right)$$

Da im Vektorraummodell sowohl Anfrage als auch Dokumente aus der Datenbasis dasselbe Format besitzen, können Ähnlichkeitsmaße leicht definiert werden. Im Kontext der Projektgruppe verwenden wir die Kosinus-Ähnlichkeit, bei der der Kosinus des Winkels zwischen zwei Vektoren ausgewertet wird. Grundlage hierfür ist der Kosinussatz.

2.2 Satz (Kosinussatz). Der Kosinus des Winkels zwischen zwei Vektoren a und b berechnet sich wie folgt:

$$\cos\left(\angle(a, b)\right) = \frac{a^T \cdot b}{\|a\| \cdot \|b\|}$$

Der Term $a^T \cdot b$ stellt hierbei das Skalarprodukt der Vektoren a und b dar. Der Operator $\|\cdot\|$ ist die euklidsche Norm.

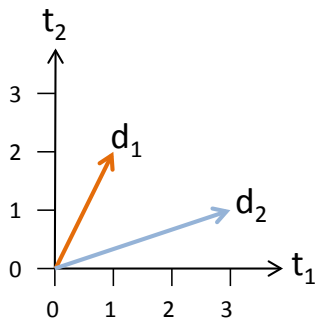


Abbildung 2.2: Ungewichtete Termfrequenz-Vektoren $d_1 = (1, 2)$ und $d_2 = (3, 1)$

Der Kosinus des von zwei Vektoren a und b aufgespannten Winkels beschreibt also die normierte Projektion von a auf b und ist daher als Ähnlichkeitsmaß für das IR geeignet. Liegen zwei Vektoren orthogonal zueinander, so teilen die zugehörigen Dokumente keine Wörter und ihre Ähnlichkeit beträgt Null. Zeigen zwei Vektoren hingegen in dieselbe Richtung, so ist die relative Verteilung sämtlicher gewichteter Wortvorkommen beider Dokumente identisch und die Ähnlichkeit ist maximal (sie hat also den Wert Eins). Im Beispiel der ungewichteten Termfrequenz-Vektoren aus Abbildung 2.2 beträgt die Kosinus-Ähnlichkeit ungefähr 0.707.

2.5 Indexstrukturen

Im Folgenden wird erläutert, aus welchem Grund wir uns bei der Entwicklung der hier vorgestellten Software für die Verwendung von Indexstrukturen entschieden haben, sowie eine genaue Beschreibung unseres Vorgehens.

Um eine effektive Erkennung von Plagiaten zu ermöglichen, benötigt eine zu diesem Zweck implementierte Software eine entsprechend große Datenbank. Diese Datenbank sollte große Mengen von wissenschaftlichen Dokumenten zu jedem Fachgebiet beinhalten, in denen die Software zum Einsatz kommen soll. Je größer die Datenmenge ist, mit der man ein zu untersuchendes Dokumente abgleichen kann, desto qualitativ hochwertiger sind die erzielten Ergebnisse. Zudem erhöht eine große Vielfalt von Fachbereichen innerhalb der Datenmenge die universelle Einsetzbarkeit der Software.

Für die Suche nach Plagiaten muss nun in allen relevanten *Kandidatendokumenten* (siehe Abschnitt 2.2) nach bestimmten Wortmustern gesucht werden. Müssten für jedes Muster des zu untersuchenden Dokumentes alle Kandidatendokumente vollständig geprüft werden, so würde die Laufzeit der Plagiatssuche asymptotisch $O(nmk)$ betragen. Hierbei ist n die Länge des zu untersuchenden Dokumentes, m die durchschnittliche Länge eines Kandidatendokumentes und k die Anzahl der Kandidatendokumente. Es wird demnach eine Technik benötigt, welche es ermöglicht die Kandidatendokumente effizienter auf ein bestimmtes Muster hin zu untersuchen. Zu diesem Zweck haben wir uns dazu entschieden eine Indexstruktur für die Kandidatendokumente und das zu untersuchende Dokument zu verwenden. Diese wird anhand der bestehenden Daten einmalig berechnet und ermöglicht es danach effizient auf bestimmte Bereiche der Datenmenge zuzugreifen, bzw. nach bestimmten Datenmustern zu suchen.

Unter den vielfältigen Typen von Indexstrukturen findet man u.a. Hashtabellen und Baumstrukturen. Wir haben uns für *Suffixarrays* entschieden [11]. Gegeben eine Zeichenkette $S = (s_1, s_2, \ldots, s_n)$ von n Zeichen. Ein Suffix S^k ist eine zusammenhängende Teilfolge von S, welche mit dem letzten Zeichen s_n von S endet, also

$$S^k = (s_k, s_{k+1}, \ldots, s_n).$$

Gegeben die Suffixe des zu Grunde liegenden Textes und deren Startindizes, so ist das Suffixarray eine Folge dieser Indizes. Die Reihenfolge der Indizes bestimmt sich durch eine gewählte Ordnung der Suffixe, in unserem Fall die natürliche Ordnung ihrer Ganzzahlen-Darstellung. Die Vorteile von Suffixarrays sind vor allem der geringe Speicherverbrauch und die Verfügbarkeit von effizienten Konstruktionsalgorithmen, welche kompatibel mit der von uns gewählten Integer-Repräsentation (siehe Abschnitt 4.3.4) für Dokumente sind. Jedoch ist die Suche nach einem bestimmten Element im Suffixarray nicht so effizient wie bei Baumstrukturen. Dieser Nachteil kann aufgrund unseres Vorgehens jedoch vernachlässigt werden, wie im Weiteren noch erläutert wird.

Als Datengrundlage haben wir die durch Konkatenation des zu untersuchenden Dokumentes und der Kandidatendokumente entstehende Zeichenkette gewählt. Dies ermöglicht

Index	0	1	2	3	4	5	6	7	8	9	10
Text	Lorem	ipsum	diam	sed	amet	consetetur	sed	elitr	sed	diam	$

Index	0	1	2	3	4	5	6	7	8	9	10
Suffixarray	10	4	5	9	2	7	1	0	3	8	6

Abbildung 2.3: Ein Beispieltext mit dem zugehörigen Suffixarray.

Index	**Suffixarray**	**Suffix**
0	10	$
1	4	amet consetetur sed elitr sed diam $
2	5	consetetur sed elitr sed diam $
3	9	diam $
4	2	diam sed amet consetetur sed elitr sed diam $
5	7	elitr sed diam $
6	1	ipsum diam sed amet consetetur sed elitr sed diam $
7	0	Lorem ipsum diam sed amet consetetur sed elitr sed diam $
8	3	sed amet consetetur sed elitr sed diam $
9	8	sed diam $
10	6	sed elitr sed diam $

Abbildung 2.4: Explizite Darstellung des Suffixarrays aus Abbildung 2.3. Es sind die, den Einträgen entsprechenden, Suffixe angegeben.

es uns in allen Kandidatendokumenten gleichzeitig nach einer Übereinstimmung mit einem Muster aus dem zu untersuchenden Dokument zu suchen. Hierbei ist daran zu erinnern, dass unsere Daten in einer Integer-Repräsentation vorliegen (siehe Abschnitt 4.3.4), was im Folgenden jedoch aus Gründen der Anschaulichkeit vernachlässigt wird. Beispielhaft wird in Abbildung 2.3 auf Basis eines Textes das entsprechende Suffixarray erstellt. Hierbei bezeichnet $ das Ende des Textes und besitzt den lexikographisch kleinsten Wert. Zu beachten ist, dass in diesem Beispiel die Indexierung wortbasiert erfolgt, so wie es auch in unserer Software der Fall ist. Das zugehörige Suffixarray ist ebenfalls in Abbildung 2.3 dargestellt. Hier wurden die Textindizes gemäß der lexikographischen Ordnung der Suffixe sortiert. Diese Liste ist für den Leser wie in Abbildung 2.4 zu interpretieren. Suchen wir nach einem bestimmten Muster im Text, so lässt sich dies auf die Suche nach einem entsprechenden Präfix im Suffixarray reduzieren. Ein Präfix wird wie folgt definiert: Sei eine Zeichenkette S ein Tupel von n Zeichen. Ein Präfix S_k ist eine zusammenhängende Teilfolge von S, welche mit dem ersten Zeichen s_1 beginnt.

$$S = (s_1, s_2, \ldots, s_n)$$
$$S_k = (s_1, s_2, \ldots, s_k)$$

Da das Suffixarray eine Ordnung bezüglich der Zahlenordnung der Integer-Repräsentation besitzt, kann mittels binärer Suche ein Eintrag mit gegebenem Präfix in logarithmischer Zeit gefunden werden. Ebenfalls kann das Fehlen eines entsprechenden Musters in der Textbasis in der gleichen Zeit festgestellt werden. Ein weiterer Vorteil des Suffixarrays erschließt sich, wenn das gesuchte Muster mehr als einmal in der Textbasis auftritt. Im oben genannten Beispiel wäre dies z.B. das Muster *sed*. Es tritt in der Textbasis an den Positionen 3, 6 und 8 auf. Aufgrund ihrer Ordnung im Suffixarray befinden sich alle Einträge

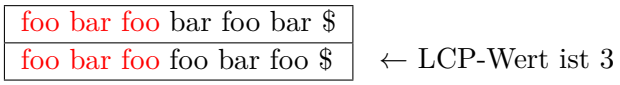


Abbildung 2.5: Beispiel für das LCP zweier Wortketten.

Index	Wortkette	LCP-Wert
0	foo bar bar bar foo bar $	-
1	foo bar foo bar foo bar $	2
2	foo bar foo foo bar foo $	3
3	foo bar foo foo foo foo $	4
4	foo foo bar bar bar foo $	1
5	foo foo foo foo bar foo $	2

Abbildung 2.6: Beispiel für die LCP-Wert-Bestimmung über größere Distanz. Die eingetragenen Werte beziehen sich auf das vorige Element. Aus diesem Grund besitzt das erste Element keinen LCP-Wert. Der LCP-Wert zwischen Element zwei und fünf ergibt sich aus dem Minimum der eingetragenen LCP-Werte der Elemente drei, vier und fünf: $lcp(2,5) = \min\{4,1,2\} = 1$.

zu Suffixen mit dem gleichen Präfix innerhalb eines Intervalls im Suffixarray (siehe die letzten drei Einträge der Abbildung 2.4). Daher können, nachdem mittels binärer Suche eines dieser Elemente gefunden wurde, mit einer linearen Suche durch die vorhergehenden und nachfolgenden Einträge alle Vorkommen des Musters identifiziert werden.

Relevant ist zudem, dass sich ein Suffixarray effizient konstruieren lässt. Zu diesem Zweck existieren inzwischen einige Algorithmen, sowie zugehörige Programmbibliotheken, welche ein Suffixarray in linearer Zeit aufbauen. Wir haben uns für den DC3-Algorithmus entschieden, da er schon in dem zugehörigen Fachartikel mit einer C++-Implementation aufwarten kann, welche leicht in unsere Software integriert werden kann und explizit auf Daten in Integer-Repräsentation ausgelegt ist [10].

Zusätzlich zum eigentlichen Suffixarray kann ein sogenanntes *LCP-Array* angelegt werden um die Suche potentiell zu beschleunigen. LCP steht für *longest common prefix*. Für zwei gegebene Strings S_1, S_2 ist S_{LCP} das längste gemeinsame Präfix von S_1 und S_2. Der LCP-Wert von S_1 und S_2 ist demnach die Länge von S_{LCP}. Der i-te Eintrag im LCP-Array beinhaltet den LCP-Wert der beiden Suffixe, welche durch die Suffixarray-Einträge i und $i-1$ referenziert werden. Seien S und T zwei Zeichenketten, so ist ihr gemeinsamer LCP-Wert:

$$\text{lcp}(S,T) = \max\{|S_i| \mid S_i = T_i, i \leq \min(|S|,|T|)\}$$

Als Beispiel besitzen die folgenden Einträge *foo bar foo foo bar foo $* und *foo bar foo bar foo bar $* einen LCP-Wert von 3, da das Präfix *foo bar foo* bei beiden Elementen zu finden ist (siehe Abbildung 2.5). Anhand der LCP-Werte benachbarter Elemente des Suffixarrays E_{i-1}, E_i kann der LCP-Wert zweier beliebiger Elemente E_j, E_k bestimmt werden, indem hierzu der niedrigste LCP-Wert aller Elemente zwischen E_j und E_k verwendet wird. Gegeben $lcp_i = lcp(E_{i-1}, E_i)$, so ist:

$$lcp(E_j, E_k) = \min\{lcp_{j+1}, lcp_{j+2}, \ldots, lcp_{k-1}, lcp_k\}$$

Dies ist in Abbildung 2.6 anhand mehrerer Wortketten verdeutlicht, indem das Beispiel aus Abbildung 2.5 um zusätzliche Einträge erweitert wurde.

Zu betrachten ist nun, inwiefern das LCP-Array zur Verbesserung der Suche im Suffixarray beitragen kann. Grundsätzlich wird davon ausgegangen, dass um einen bestimmten Element im Suffixarray zu suchen ein binäres Suchverfahren verwendet wird. Hierbei wird stets ein einzelner Eintrag genauer betrachtet. Dieses wird als aktuelles Element S_{current} bezeichnet. Gesucht wird ein Eintrag im Suffixarray welcher mit einem gegebenen Präfix-Muster S_{pattern} beginnt. Hierbei besteht durchaus die Möglichkeit, dass ein solches Element gar nicht oder sogar mehrfach im Suffixarray vorhanden ist. Beides stellt kein Problem dar, da im ersten Fall das Fehlen des Elementes festgestellt wird, sobald das Suchfenster auf ein einzelnes Element beschränkt ist und im zweiten Fall lediglich irgendein passender Eintrag als vorläufiges Ergebnis ausreicht. Nachdem die Übereinstimmung von S_{current} und S_{pattern} überprüft und für nicht vollständig befunden wurde, kann anhand der Ordnung des Suffixarrays bestimmt werden, ob sich das gesuchte Element im Intervall vor oder hinter S_{current} befinden muss, sofern es existiert. Das mittlere Element dieses Intervalls wird im nächsten Schritt untersucht werden und im aktuellen Durchgang als nächstes Element S_{next} bezeichnet. Bei der Suche können nun die LCP-Werte zwischen S_{pattern} und S_{current}, definiert als $\text{lcp}(S_{\text{pattern}}, S_{\text{current}})$ (Berechnung analog zu Abbildung 2.6), sowie S_{next} und S_{current}, definiert als $\text{lcp}(S_{\text{next}}, S_{\text{current}})$ (siehe Abbildung 2.6), verglichen werden. Da der LCP-Wert zwischen zwei Elementen wie oben beschrieben dem Minimum aller LCP-Werte der dazwischen liegenden Elemente entspricht, nimmt der LCP-Wert bei zunehmender Distanz der Elemente monoton ab. Gesucht wird jenes Element, bei dem $\text{lcp}(S_{\text{pattern}}, S_{\text{current}})$ der Länge des Musters entspricht. In den folgenden Fällen wird davon ausgegangen, dass dieses Element noch nicht identifiziert wurde und die binäre Suche im Suffixarray weitergeführt werden muss. Es gilt $\text{lcp}(S_{\text{pattern}}, S_{\text{current}}) < |S_{\text{pattern}}|$ und anhand des Vergleichs zwischen S_{pattern} und S_{current} ist bekannt ob die Suche vor oder hinter S_{current} fortgesetzt werden muss. Hieraus ergibt sich die Wahl für das nächste zu untersuchende Element S_{next}. Zu Entscheiden ist nun ob S_{pattern} mit dem S_{next} tatsächlich verglichen werden muss.

Fall 1: $\text{lcp}(S_{\mathbf{pattern}}, S_{\mathbf{current}}) > \text{lcp}(S_{\mathbf{next}}, S_{\mathbf{current}})$**.** Der LCP-Wert fällt auf dem Weg von S_{current} zu S_{next} ab. Da der LCP-Wert zum Muster jedoch auf dem Weg von S_{current} zum gesuchten Element monoton steigen muss, kann auf diese weise gesagt werden, dass sich S_{next} bereits hinter dem gesuchten Element befindet. Somit kann ohne Wortvergleich gesagt werden, dass das neue Suchintervall zwischen S_{current} und S_{next} liegen muss. Zu beachten ist, dass sich S_{current} hierbei nun nicht ändert, sondern lediglich S_{next} das mittlere Element im neuen Suchintervall zugewiesen wird. Abbildung 2.7 verdeutlicht diesen Fall.

Fall 2: $\text{lcp}(S_{\mathbf{pattern}}, S_{\mathbf{current}}) < \text{lcp}(S_{\mathbf{next}}, S_{\mathbf{current}})$**.** In diesem Fall kann ähnlich zu Fall 1 vorgegangen werden. Da $\text{lcp}(S_{\text{next}}, S_{\text{current}})$ größer ist als $\text{lcp}(S_{\text{pattern}}, S_{\text{current}})$ hat S_{next} ein längeres Präfix mit S_{current} gemein als mit S_{pattern}. Dies bedeutet, dass ein direkter Vergleich von S_{pattern} und S_{next} an der gleichen Stelle des Musters keine Übereinstimmung mehr erkennen würde, wie bei S_{current} und S_{pattern}. Demnach kann das gesuchte Element nicht in dem Intervall zwischen S_{current} und S_{next} liegen. Die Zuweisung von S_{next} wird auf das nächste mittlere Element im Intervall hinter S_{next} festgelegt. Abbildung 2.8 verdeutlicht diesen Fall.

Fall 3: $\text{lcp}(S_{\mathbf{pattern}}, S_{\mathbf{current}}) = \text{lcp}(S_{\mathbf{next}}, S_{\mathbf{current}})$**.** Im letzten Fall kann anhand der Werte von $\text{lcp}(S_{\text{next}}, S_{\text{current}})$ und $\text{lcp}(S_{\text{pattern}}, S_{\text{current}})$ keine Aussage über die Position des gesuchten Elements gemacht werden. In diesem Fall muss ein direkter Vergleich der weiteren Zeichen von S_{pattern} und S_{next} erfolgen um das darauf folgende Element der binären Suche zu bestimmen. Abbildung 2.9 verdeutlicht diesen Fall. Mit die-

ser Technik wird sichergestellt, dass alle Zeichen des Musters nur einmal verglichen werden, was die Suche somit beschleunigt.

$S_{\text{pattern}} = \{\text{foo}, \text{bar}, \text{foo}, \text{bar}, \text{goo}\}$

Index	Wortkette	LCP-Wert	
0	foo bar bar bar foo bar $	-	
1	foo bar foo bar foo bar $	2	$\leftarrow S_{\text{current}}$
2	foo bar foo foo bar foo $	3	
3	foo bar foo foo foo foo $	4	$\leftarrow S_{\text{next}}$
4	foo foo bar bar bar foo $	1	
5	foo foo foo foo bar foo $	2	

Abbildung 2.7: Suche im Array aus Abbildung 2.6 für $\text{lcp}(S_{\text{pattern}}, S_{\text{current}}) > \text{lcp}(S_{\text{next}}, S_{\text{current}})$. Es gilt: $\text{lcp}(S_{\text{pattern}}, S_{\text{current}}) = 4$ und $\text{lcp}(S_{\text{next}}, S_{\text{current}}) = 3$. Wenn das gesuchte Muster S_{pattern} demnach im Array vorhanden ist, so muss es sich zwischen S_{current} und S_{next} befinden.

$S_{\text{pattern}} = \{\text{foo}, \text{bar}, \text{goo}\}$

Index	Wortkette	LCP-Wert	
0	foo bar bar bar foo bar $	-	
1	foo bar foo bar foo bar $	2	$\leftarrow S_{\text{current}}$
2	foo bar foo foo bar foo $	3	
3	foo bar foo foo foo foo $	4	$\leftarrow S_{\text{next}}$
4	foo foo bar bar bar foo $	1	
5	foo foo foo foo bar foo $	2	

Abbildung 2.8: Suche im Array aus Abbildung 2.6 für $\text{lcp}(S_{\text{pattern}}, S_{\text{current}}) < \text{lcp}(S_{\text{next}}, S_{\text{current}})$. Es gilt: $\text{lcp}(S_{\text{pattern}}, S_{\text{current}}) = 2$ und $\text{lcp}(S_{\text{next}}, S_{\text{current}}) = 3$. Wenn das gesuchte Muster S_{pattern} demnach im Array vorhanden ist, so muss es sich hinter S_{next} befinden.

$S_{\text{pattern}} = \{\text{foo}, \text{bar}, \text{foo}, \text{goo}\}$

Index	Wortkette	LCP-Wert	
0	foo bar bar bar foo bar $	-	
1	foo bar foo bar foo bar $	2	$\leftarrow S_{\text{current}}$
2	foo bar foo foo bar foo $	3	
3	foo bar foo foo foo foo $	4	$\leftarrow S_{\text{next}}$
4	foo foo bar bar bar foo $	1	
5	foo foo foo foo bar foo $	2	

Abbildung 2.9: Suche im Array aus Abbildung 2.6 für $\text{lcp}(S_{\text{pattern}}, S_{\text{current}}) = \text{lcp}(S_{\text{next}}, S_{\text{current}})$. Es gilt: $\text{lcp}(S_{\text{pattern}}, S_{\text{current}}) = 3$ und $\text{lcp}(S_{\text{next}}, S_{\text{current}}) = 3$. Unter diesen Umständen kann keine Aussage darüber gemacht werden ob das gesuchte Muster S_{pattern} vor oder hinter S_{next} liegt, oder gar mit dem Präfix von S_{next} übereinstimmt.

KAPITEL 3

Bestehende Lösungsansätze

In diesem Kapitel gehen wir auf bereits bestehende Lösungsansätze zum Finden von Plagiaten ein. Einerseits beschreiben wir kommerzielle Lösungen, vor allem für den akademischen Bereich, andererseits befassen wir uns mit dem PAN-Wettbewerb, der in einer Teilaufgabe das Auffinden von kopierten Textstellen zum Ziel hatte.

Die Qualität der Ergebnisse einer Plagiatserkennungssoftware ist auch von der Qualifikation des Benutzers abhängig. Das bedeutet, dass der Benutzer sowohl die Bedienung des Programms beherrscht, als auch in das Thema des zu überprüfenden Dokuments eingeweiht ist. Resultate von Plagiatserkennungssoftware müssen immer überprüft werden, da korrekt zitierte Textstellen oder Standardformulierungen als Plagiat erkannt werden können, siehe dazu auch Abschnitt 2.2.

3.1 Kommerzielle Lösungen

Zahlreiche Hersteller haben Produkte entwickelt, die versprechen, Plagiate in Texten zu finden. Durch einen Software-Lizenzvertrag der TU Dortmund erhielten wir Zugriff auf die kommerziellen Produkte *Turnitin* und *Ephorus*.

Da bei dieser Form der Plagiatssuche Dokumente auf Server eines Anbieters hochgeladen werden, ist das Urheberrecht zu beachten. Das bedeutet, dass der Urheber des Dokuments die Nutzungsrechte an dem entsprechenden Dokument der entsprechenden Firma einräumen muss. Wird dieses Recht nicht eingeräumt, stehen dem Urheber, wie in Abschnitt 2.1 erwähnt, verschiedene Rechte zu. Zusätzlich ist denkbar, dass Arbeiten verschiedene persönliche Daten wie Namen, Matrikelnummer oder (Email-)Adresse erhalten. Dann ist zusätzlich das Bundesdatenschutzgesetz (BDSG) zu beachten, welches sehr enge Grenzen für die Nutzung solcher Daten vorsieht.

3.1.1 Turnitin

Turnitin ist ein Softwareprodukt der Firma iParadigms[1]. Gegründet wurde das Projekt von vier Absolventen der UC Berkeley, um ein System für Peer-Review-Anträge zu programmieren. Mittlerweile ist Turnitin ein Service, der täglich 300.000 Arbeiten auf Plagiate überprüft, an Spitzentagen sogar bis zu 600.000. Turnitin unterstützt 19 verschiedene

[1] `http://www.turnitin.com/` (Zuletzt abgerufen: 06.04.2016)

Sprachen, und im Archiv sind 45 Millionen Webseiten, 400 Millionen Arbeiten von Studenten und 130 Millionen Artikel aus über 110.000 Journalen und Büchern gespeichert. Turnitin bietet neben der Plagiatsüberprüfung auch eine Plattform für die Korrektur und Bewertung von studentischen Arbeiten.

Um ein Dokument zu überprüfen, wird dieses über ein Webinterface auf den Server von Turnitin hochgeladen. Dazu wird erst ein Kurs erstellt, für den das Dokument eingereicht werden soll, und eine Aufgabe in diesem Kurs. So kann der Dozent eingereichte Arbeiten sortieren. Zusätzlich kann ein Link für eine Aufgabe erzeugt werden, der weitergegeben werden kann, damit Studenten selbstständig Arbeiten hochladen können. Beim Hochladen lassen sich drei verschiedene Ablagen wählen. In der Standardablage, die voreingestellt ist, werden alle hochgeladenen Dokumente aller Nutzer in eine einzige Datenbank eingefügt. Alle Dokumente werden bei einer Überprüfung mit dieser Datenbank verglichen. So können Plagiate von Arbeiten festgestellt werden, die zu einem früheren Zeitpunkt bereits bei Turnitin hochgeladen wurden, aber sonst nicht auffindbar sind. Wird die *Arbeitsablage Institution* gewählt, so wird das Dokument in eine separate Datenbank abgelegt, die nur von der entsprechenden Institution genutzt werden kann. Alle Dokumente dieser Institution werden auch mit Dokumenten in dieser Datenbank verglichen, aber Arbeiten anderer Institutionen nicht. Letztlich kann auch keine Ablage gewählt werden. Bei dieser Option wird das Dokument in keine Datenbank eingefügt und wird bei zukünftigen Plagiatssuchen nicht berücksichtigt. Trotzdem ist das Dokument von dem jeweiligen Benutzer jederzeit abrufbar. Weitere Parameter, beispielsweise die minimale Länge eines Plagiats, sind nicht verfügbar.

Nach einiger Zeit, abhängig von der Länge des Dokuments, steht ein Ergebnis der Plagiatsprüfung zur Verfügung. Meist dauert dies nur wenige Minuten, bei langen Dokumenten bis zu einigen Stunden. Wird das Ergebnis aufgerufen, wird das hochgeladene Dokument mit allen Formatierungen und Bildern angezeigt. Plagiierte Stellen werden in verschiedenen Farben hervorgehoben und nummeriert. Die Quelle wird in gleicher Farbe und entsprechender Nummerierung auf der rechten Seite angezeigt. Per Mausklick auf einen Pfeil neben der Quelle kann man zu dem Originaltext gelangen.

Zum Testen haben wir ein Testplagiat angefertigt, welches aus der deutschen Wikipedia-Seite von Karl Theodor von und zu Guttenberg, einer aktuellen Nachricht von dem Nachrichtenportal `www.heise.de` und einem Abschnitt aus einem online als pdf-Dokument einsehbaren Buch [12] besteht.

Die Textstellen wurden unverändert übernommen und nicht als Zitate gekennzeichnet. Jede der drei Textstellen ist jeweils etwa eine halbe Seite lang. Von den drei Quellen wurde nur Wikipedia erkannt und komplett gekennzeichnet. Die beiden anderen Quellen tauchen nicht auf. Dafür findet Turnitin zwei weitere Quellen in der Textstelle des übernommenen Buches, die in wenigen Worten mit dem plagiierten Text übereinstimmen. Turnitin kommt in dem Prüfungsbericht zu dem Ergebnis, dass 33% des Textes plagiiert sind.

Für einen zweiten Test haben wir von SCIgen[2] ein Paper automatisch generieren lassen. SCIgen ist ein Programm, welches automatisiert Paper aus dem Bereich der Informatik erzeugt. Dabei wird eine kontextfreie Grammatik verwendet, die von den Autoren des Programms erstellt wurde. Das so erzeugte Dokument hat ein ähnliches Format und Aufbau wie wissenschaftliche Texte aus Zeitschriften. Dies bedeutet, es enthält einen Titel, Abstract, Textkörper mit Literaturverweisen und ein Literaturverzeichnis. Die angegebenen Quellen sind dabei jedoch auch generiert. Turnitin kam zu den Ergebnis, das 52% des Textes plagiiert sind. Dabei stellt Turnitin 58 Quellen fest. Die markierten Stellen sind höchstens ein Satz lang und bestehen aus Standardformulierungen. Neben den als

[2] `https://pdos.csail.mit.edu/archive/scigen/`

Plagiat markierten Stellen im eigentlichen Text werden auch Stellen im Inhaltsverzeichnis, Titelblatt oder Literaturverzeichnis markiert. Beide Testdokumente sind im Anhang angefügt.

3.1.2 Ephorus

Bei Ephorus[3] handelt es sich um eine weitere kommerzielle Plagiatserkennungssoftware. Mittlerweile wurde diese von Turnitin aufgekauft. Der Funktionsumfang beschränkt sich hier lediglich auf die Plagiatserkennung. Hochladen lassen sich Dokumente vom Typ Word, PDF, Textdokumente, Open-Office-Text und Internetseiten mit einer maximalen Größe von 25 MB. Darüber hinaus lassen sich Zip-Archive mit maximal 8 MB hochladen. Im Gegensatz zu Turnitin muss zum Hochladen der Plagiate kein Kurs und keine Aufgabe erstellt werden, zum Sortieren lassen sich aber dennoch Ordner anlegen und beim Hochladen auswählen. Darüber hinaus lassen sich ähnlich wie bei Turnitin Abgabecodes erzeugen, damit Studenten ihre Abgaben direkt in einen Ordner des Lehrenden, also dem Benutzer von Ephorus, hochladen können.

Zusätzlich kann beim Hochladen die Art der Prüfung ausgewählt werden. Zur Auswahl stehen Standardprüfung, Referenzmaterial und private Überprüfung. Bei der Standardprüfung wird das Dokument einer Plagiatsprüfung unterzogen und als zukünftige Quelle für Plagiate behandelt. Bei der Prüfung als Referenzmaterial wird das Dokument keiner Plagiatsüberprüfung unterzogen, aber es wird als Referenzmaterial für zukünftige Überprüfungen herangezogen. Bei einer privaten Prüfung wird das hochgeladene Dokument auf Plagiate untersucht, aber nicht als Referenzmaterial für zukünftige Prüfungen verwendet.

Nach einiger Zeit steht der Bericht zur Verfügung. In der Übersicht der Dokumente wird dabei der Name der hochgeladenen Datei und eine Prozentzahl angezeigt, welche für den Anteil an Plagiaten im Dokument steht. Wird der Bericht aufgerufen, erscheint ein Bericht über Plagiate in dem Dokument. Dieser Bericht steht in zwei Varianten zur Verfügung. In der Variante *Zusammenfassung* werden alle gefunden Stellen markiert. Es lassen sich per Auswahl einzelne Quellen vom Bericht ausschließen. In der alternativen Variante *Detailliert* ist lediglich eine Quelle gleichzeitig ausgewählt. Die Texte des Plagiatdokuments und des Quelldokuments werden dabei nebeneinander dargestellt. Die Quellen lassen sich in beiden Varianten in der Quellliste auswählen.

Für den Bericht lassen sich drei Stufen aufrufen: *Standard*, *Nachgiebig* und *Streng*. Diese Auswahl beeinflusst die minimale Länge von verdächtigen Stellen, und damit indirekt auch die Anzahl der Quellen.

Der Text des Dokuments folgt auf die Quellliste. In Ephorus gehen alle Formatierungen und Grafiken verloren, es wird lediglich der reine Text angezeigt, was den Bericht unübersichtlich gestaltet. Verdächtige Stellen werden mit rotem Text dargestellt. Wörter in blauer Schrift in einer verdächtigen Stelle weisen auf Abweichungen vom Original hin. Laut Handbuch werden mögliche Zitate mit blau unterstrichen, dies haben wir jedoch nicht überprüft. Bei einem Test mit der Bachelorarbeit einer unserer Mitautoren haben wir festgestellt, das Zitate rot unterstrichen werden. Da die Texterkennung von Ephorus nicht mit deutschen Anführungszeichen umgehen kann, sind teilweise Stellen als Zitat markiert, die mehrere Absätze umfassen. Sobald der Mauszeiger über einer markierten Textstellen schwebt, wird die zugehörige Quelle als URL in einem Tooltip angezeigt. Die Quelle lässt sich trotzdem lediglich über einen Klick in der Quellenübersicht aufrufen.

Beim Test der Plagiatserkennung wurden die selben Dokumente als Eingabe verwendet, wie sie für Turnitin verwendet wurden. Bei dem selbst angefertigten Testplagiat wurden zwar alle Quellen richtig erkannt, aber der übernommene Text nicht komplett markiert.

[3] `https://www1.ephorus.com/` (Zuletzt abgerufen: 06.04.2016)

Somit kommt Ephorus auf einen Plagiatsanteil von 22%. Die nachgiebige Prüfung kommt auf einen Anteil von 11%, die strenge Prüfung ergibt einen Plagiatsanteil von 66%, es werden aber zusätzlich sechs falsche Quellen gefunden.

Bei der Prüfung des generierten Papers werden Plagiate in 20% des Dokuments festgestellt, wobei neun unterschiedliche Quellen gefunden wurden. Bei der nachgiebigen Prüfung werden lediglich 4% des Dokuments als Plagiat von vier verschiedenen Quellen erkannt. Wird das Dokument einer strengen Prüfung unterzogen, werden schon 49% des Dokuments als Plagiat aus zehn verschiedenen Quellen erkannt. Lediglich bei der strengen Prüfung wird das Literaturverzeichnis teilweise als Plagiat markiert.

3.1.3 Test von Plagiatssoftware

Eine Übersicht[4] über existierende Plagiatssoftware wurde von Prof. Dr. Debora Weber-Wulff an der Hochschule für Technik und Wirtschaft Berlin angefertigt. Dabei wurden verschiedene Softwareprodukte auch bezüglich ihrer Funktionsweise gegeneinander getestet. Da diese Systeme kommerziell sind, ist nichts über die Arbeitsweise dieser Systeme bekannt. Turnitin wurde zusammen mit *Urkund* und *Copyscape* in die Kategorie der teilweise nützlichen Systeme eingestuft, die Kategorie der Plagiatsscanner mit den besten Ergebnissen. Ephorus wurde als begrenzt nützliches System eingestuft, und einige Produkte sind laut dem Test für akademische Zwecke nicht geeignet. Eine Übersicht über die Ergebnisse des Tests von 2013 liefert Tabelle 3.1.

Das Plagiatsscanner Plagiate nicht immer als solche erkennen, wurde ebenfalls von Prof. Weber-Wulff gezeigt. Dabei wurde die Doktorarbeit von Karl-Theodor von und zu Guttenberg mit fünf verschiedenen Plagiatsscannern überprüft [24]. Diese Doktorarbeit wurde von dem Projekt GuttenPlag[5] überprüft. Dort wurde festgestellt, dass auf 94% der Seiten oder 63% der Zeilen ein Plagiat zu finden ist. Das Programm *PlagAware* scheiterte an der Länge des Textes, konnte aber auf den ersten 159 Seiten 68% an Plagiaten feststellen. *iThenticate* bewertete die Arbeit mit 40% Plagiat, *PlagScan* mit 15,9% und Urkund mit 21%. Ephorus dagegen erkannte in der Arbeit dagegen nur 5% als plagiiert, was unter der von Ephorus definierten Schwelle einer plagiierten Arbeit fällt und somit als nicht plagiiert gekennzeichnet wird.

3.2 PAN-Wettbewerb

Der PAN-Wettbewerb[6] ist ein jährlich stattfindender Wettbewerb, dessen Ziel die effiziente Evaluierung von Plagiarismus, Autorenschaft und Missbrauch von sozialen Medien ist. Der PAN-Wettbewerb fand im Jahr 2015 zum 13. mal im Rahmen der CLEF-Konferenz statt, welche in dem Jahr vom 8. bis zum 11. September in Toulouse, Frankreich abgehalten wurde. Auch im Jahr 2016 wird der PAN-Wettbewerb wieder ausgerichtet, dieses mal in Évora, Portugal.

3.2.1 Aufgaben

Zur Teilnahme an vergangenen Wettbewerben musste eine von drei verschiedenen Aufgaben bearbeitet werden. Bei der Detektierung von Plagiarismus sollten in verdächtigen Dokumenten verdächtige Stellen gefunden werden. Die Aufgabe der Autorenidentifikation

[4] `http://plagiat.htw-berlin.de/` (Zuletzt abgerufen: 06.04.2016)
[5] `http://de.guttenplag.wikia.com/wiki/GuttenPlag_Wiki` (Zuletzt abgerufen: 06.04.2016)
[6] `http://pan.webis.de/` (Zuletzt abgerufen: 06.04.2016)

Software	Effektivität	Benutzbarkeit gemäß Checkliste	Benutzbarkeit subjektiv
Urkund	95	12,5	10
Turnitin	87	15,5	12
Copyscape	87	15	7
Ephorus	76	19	9
PlagAware	75	19	11
Strike Plagiarism	75	17	10
PlagScan	72	17	9
Compilatio	72	15	4
PlagiarismDetectPremium	72	12	5
Docoloc	70	13	4
PlagiarismDetectStandard	65	12	5
Duplichecker	63	12	5
PlagTracker	41	12	7
Plagiarisma	39	7	2
OAPS	39	11	6
PlagiarismFinder	38	19	11

Tabelle 3.1: Test von Plagiatsscannern von Prof. Weber-Wulff aus dem Jahr 2013. Maximale Effektivität ist 130 Punkte, auf der Checkliste Benutzbarkeit konnten 27 Punkte erzielt werden und bei der subjektiven Bewertung der Benutzbarkeit maximal 15 Punkte.

war, bei zwei verschiedenen Dokumenten festzustellen, ob diese vom selben Autor geschrieben wurden. Die dritte Aufgabe war, ein Profil des Autors eines Texts zu erstellen. Für uns war vor allem die erste Aufgabe von Interesse. Aber auch die Ergebnisse der anderen beiden Aufgaben können Rückschlüsse auf Plagiate geben, z.B. kann das Profil des Autors auch als Ähnlichkeitsmaß für Plagiate dienen. Ab dem Jahr 2016 verändern sich die verschiedenen Aufgabenstellungen. Die Detektierung von Plagiaten wird aus dem Programm genommen, dafür kommt als neue Aufgabe hinzu, den Autor eines Textes zu verschleiern.

Die Detektierung von Plagiarismus teilte sich in zwei Teilaufgaben auf, die unabhängig voneinander gelöst werden konnten. Die erste Aufgabe war *Source Retrieval*, in der für ein gegebenes verdächtiges Dokument Quellen gesucht wurden, aus denen kopiert worden sein könnte. Da dort eine dynamische Datenbasis mitsamt Suchmaschine vorgegeben wurde, ist dies nicht zu unserem Lösungsansatz kompatibel, bei dem ein Index über eine statische Datenbasis aufgebaut wird.

Die zweite Aufgabe war das *Text Alignment* mit vorgegebenem Kandidatendokumenten. Die Vorgehensweise des Text Alignments unterteilen die Veranstalter des PAN-Wettbewerbs in drei Schritte. Die Software jeden Teilnehmers lässt sich darin einteilen. Im ersten Schritt, dem sogenannten Seeding, werden Übereinstimmungen zwischen Plagiat und Quelle gesucht. Darauf folgt die Extension, welches das Verknüpfen von Übereinstimmungen zur maximalen Länge bezeichnet. Im letzten Schritt wird im Filtering Passagen gefiltert, die bestimmte Kriterien nicht erfüllen, z.B. zu kurze Passagen oder Passagen, die dem Original nicht ähnlich genug sind. Zusätzlich verwendet jeder Ansatz verschiedene Techniken zur Vorverarbeitung von Texten.

Die Unterteilung in diese Schritte lässt sich auch bei uns vornehmen. Die Aufgabe des Seeding übernimmt bei unseren Ansatz die Suffixarrays. Extension geschieht im Postprocessor, genauso wie das Filtering.

Bewertet werden Einsendungen nach den Maßen *Precision*, *Recall* und *Granularität*. Precision ist der Anteil der relevanten Plagiatsfälle. Der Anteil der gefundenen relevanten

Plagiatsfälle nennt sich Recall. Das Maß Granularität zeigt an, wie oft ein Plagiatsfall gefunden wurde. Dies bedeutet, dass Plagiatsfälle, die mehrmals gefunden und als Plagiat angezeigt werden, negativ in die Bewertung einfließen. Für die Evaluation unserer eigenen Software in Kapitel 7 haben wir uns auf die Maße Precision und Recall beschränkt. Dort findet sich auch eine genauere Beschreibung von Precision und Recall.

3.2.2 Ergebnisse

Der Gewinner des PAN-Wettbewerbs 2014 [17] ist eine Forschergruppe um Miguel A. Sanchez-Perez des Instituto Politécnico Nacional [20]. Der Lösungsansatz dieser Gruppe basiert auf dem tf-idf-Maß, der bei uns lediglich für die Vorauswahl der Dokumente verwendet wird. Die Grundidee des Algorithmus dieser Gruppe ist, dass nicht komplette Dokumente als Vektor repräsentiert werden, sondern jeder einzelne Satz als Dokument betrachtet wird.

Im Schritt der Vorverarbeitung wurden alle Wörter auf ihren Stamm reduziert. Darüber hinaus wurden die Texte anhand ihrer Satzzeichen zerlegt, und bei sehr kurzen Sätzen mit dem nächsten verknüpft. Zusätzlich wurden alle Wörter in die Kleinschreibung überführt und Wörter, die nicht mit einem Buchstaben oder einer Zahl beginnen, entfernt. Ähnliche Sätze wurden mit zwei verschiedenen Ähnlichkeitsmaßen gesucht. Neben der Kosinus-Ähnlichkeit, wie sie in unserem Ansatz verwendet wird, wird auch die Dice-Ähnlichkeit verwendet. Wenn beide Ähnlichkeitsmaße über den vorher definierten Schwellwert (Initial 0,33 für beide Maße) geraten, wird der Satz aus dem verdächtigen Dokument als Plagiat markiert und die Quelle vermerkt. Darauf folgend wurden nah beieinander liegende verdächtige Stellen miteinander zu einer verdächtigen Stelle verknüpft. Sollte es so am Ende zu Überlappungen kommen, wird die Passage genommen, die die höchste Ähnlichkeit mit dem Original aufweist. Zum Schluss werden alle Ergebnisse gefiltert, deren Länge nicht über eine Mindestlänge kommt, ähnlich zu unserem Ansatz.

Dicht gefolgt wird dieser Ansatz von Oberreuter und Eiselt [13]. Diese verwenden einen modifizierten BLAST-Algorithmus. *BLAST* (Basic Local Alignment Search Tool) ist ein in der Bioinformatik verwendetes Verfahren, um mit hoher Präzision und geringem Laufzeitaufwand neue Proteine (Sequenzen von Aminosäuren) mit bereits bekannten Proteinen einer riesigen Datenbasis zu vergleichen, indem lokale Ähnlichkeiten durch Untersuchung von Teilsequenzen gesucht werden [1]. Eine genauere Analyse ist uns nicht möglich, da wir keine Arbeiten finden können, die den verwendeten Algorithmus genauer beschreibt. Weitere Vorgehensweisen lassen sich in der Übersicht des PAN-Wettbewerbs von 2014 finden [17].

KAPITEL 4

Lösungsansatz der PG

In diesem Kapitel wird der Ansatz näher vorgestellt, den unsere Gruppe für die Plagiatssoftware SciencePlag umsetzt. Zuerst gehen wir in Abschnitt 4.1 auf die allgemeinen Ideen ein und stellen danach den Zusammenhang zwischen den einzelnen Komponenten dar. Diese Komponenten werden dann in den anschließenden Abschnitten dieses Kapitel im Detail durchgegangen.

4.1 Aufbau der Software

Für uns war es von großer Wichtigkeit, eine modulare Software zu entwickeln. Zur Erkennung von Plagiaten wird eine sogenannte *Plagiatserkennungs-Pipeline* durchlaufen. Das ist eine Folge von einzelnen, voneinander getrennten Schritten, an deren Ende das Resultat der Plagiatsfindung steht. Jeder dieser Schritte kann auf viele unterschiedliche Arten realisiert werden und so sollte jeder von ihnen ein unabhängiges Modul bilden, welches einfach ausgetauscht werden kann, ohne Änderungen in anderen Modulen notwendig zu machen. Die Qualität des Resultats der Plagiatsfindung hängt nicht nur von der Wahl der eingesetzten Algorithmen ab, sondern auch von deren Parametrisierung. Unser Plan war es daher, eine vielfältig parametrisierbare Software zu entwickeln und im Anschluss durch das systematische Testen verschiedener Parameterwerte festzustellen, welche Einstellungen besonders gut zur Erkennung von Plagiaten geeignet sind. Diesen beiden Anforderungen sind wir mit der Erstellung des Kommandozeilenprogramms SciencePlag nachgekommen.

Eine weitere Anforderung dafür, dass eine Plagiatssoftware in der Praxis Verwendung findet, ist eine einfache Bedienbarkeit über eine graphische Benutzerschnittstelle. Insbesondere eine intuitive und schnell verständliche Darstellung der Ergebnisse der Plagiatssuche ist wichtig, damit der Nutzer verdächtige Stellen zügig überprüfen kann. Zu diesem Zweck

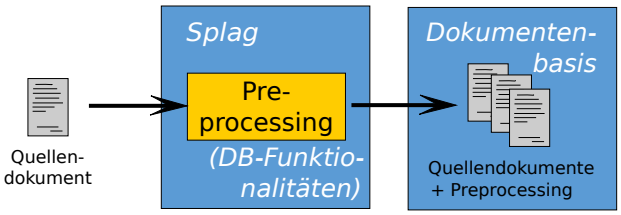


Abbildung 4.1: Verwendung von SciencePlag zur Erstellung einer Datenbasis.

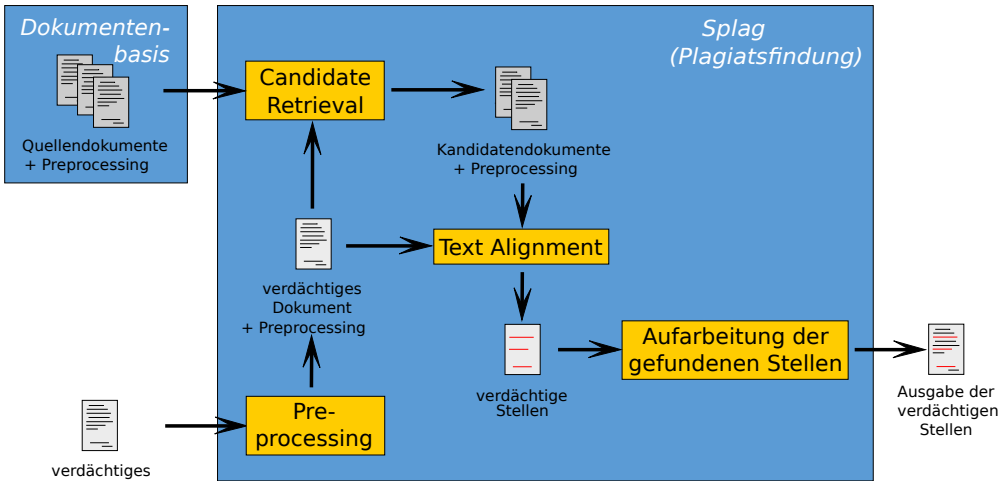


Abbildung 4.2: Verwendung von SciencePlag zur Plagiatsfindung. Datenbasis bereits vorhanden.

haben wir zusätzlich *Output Viewer* entwickelt, mit dem die Resultate von SciencePlag so aufbereitet werden, dass sie in einem Internetbrowser bequem angeschaut werden können, siehe Abschnitt 4.6.2.

Das Programm SciencePlag hat zwei Aufgaben: die Verwaltung von statischen Dokumentenbasen (Abbildung 4.1) und die Plagiatssuche an sich (Abbildung 4.2). Wir haben SciencePlag als eine Plagiatsfindungssoftware mit statischer Dokumentenbasis realisiert. Dies gibt uns Möglichkeiten zur Vorverarbeitung der Texte, die die eigentliche Plagiatsuntersuchung beschleunigen und die Resultate verbessern. Die aus mehreren Quelldokumenten erzeugten Dokumentenbasen sind dabei nicht an SciencePlag gekoppelt, insbesondere können mehrere von ihnen erstellt und verwaltet werden, um z. B. verschiedene Sprachen zu unterstützen. In einer Dokumentenbasis sind zusätzlich die Information aus der Vorverarbeitung der eingelesenen Quelldokumente gespeichert, die während des *Preprocessings* generiert wurden (Informationen über die Dokumentenbasis finden sich in Abschnitt 4.2).

Um ein verdächtiges Dokument auf Plagiatsstellen zu überprüfen, führt SciencePlag zunächst die gleichen Preprocessingschritte darauf aus, die auch auf sämtliche Quelldokumente angewandt wurden. Insbesondere werden die einzelnen Wörter in das *Integer-Alphabet* der Dokumentenbasis übersetzt, in dem jedes distinkte Ursprungswort durch eine Ganzzahl repräsentiert wird. Durch den Candidate Retrieval wird anschließend bestimmt, aus welchen Dokumenten in der verwendeten Dokumentenbasis plagiiert worden sein könnte. Auf dieser gefilterten Menge der Kandidatendokumente arbeitet nun das Text Alignment. Dort wird dann auf Basis der bereits gefundenen Informationen nach plagiierten Stellen gesucht. Da letztere danach noch in einer internen, nicht direkt verwendbaren Präsentation vorliegen, werden sie zuletzt in einem Aufbereitungsschritt in eine Form gebracht, in der sie dann evtl. noch mit einem Hilfsprogramm wie *Output Viewer* weiterverarbeitet werden können.

Details zur Software SciencePlag, insbesondere zur Verwendung, finden sich in Kapitel 5. Im Folgenden werden die Komponenten genauer beschrieben.

4.2 Dokumentenbasis

Ziel unserer Umsetzung der Dokumentenbasis ist es, große Mengen von Quelldokumenten effizient verarbeiten zu können. Angenommen, ein zufällig gewähltes Dokument enthält

tatsächlich plagiierte Textstellen. Ein Erfolg bei der Plagiatssuche, d.h. die plagiierten Stellen sowie die Dokumente aus denen sie entnommen wurden, zu finden, ist inhärent mit der Größe und Vielseitigkeit der verwendeten Dokumentenbasis verknüpft. Eine Dokumentenbasis ist somit meist sehr groß. Eine mögliche Quelle für Plagiatoren wegen der großen Anzahl an einfach zu erreichenden Artikeln ist zum Beispiel das Online-Lexikon Wikipedia[1]. Eine universell einsetzbare Plagiatserkennung sollte daher deren Artikel in ihrer Dokumentenbasis beinhalten. Die deutsche Wikipedia umfasst circa 1.9 Millionen vollständige Artikel[2]. Das vollständige Abspeichern von all diesen erfordert ca. 20 GB Speicherplatz (sie werden zum Download in einem speziellen XML-Format bereitgestellt[3]). In der englischen Sprache werden diese Zahlen mit knapp 5 Millionen noch deutlich übertroffen. Es ist auf einem Heimrechner also noch nicht üblich, dass diese Datenmengen vollständig im Hauptspeicher belassen werden können. Die Umsetzung einer Dokumentenbasis erfordert also effiziente Datenstrukturen, sowohl betreffend der Zugriffsgeschwindigkeit, als auch der Speichereffizienz.

4.2.1 Aufbau der gespeicherten Dokumente

Im Laufe unseres Plagiatserkennungsprozesses benötigen wir mehrere verschiedene Repräsentationen eines Dokumentes. Wir verwenden eine Datenstruktur `Document` um diese alle zu bündeln. Die Repräsentationen sind:

1. Der durch den Parser aus der Originaldatei extrahierte **Originaltext** des Dokuments. Dieser wird offensichtlich dazu benötigt, den vorverarbeiteten Text zu generieren. Zum anderen wird er dafür genutzt, um verdächtige Stellen für den Nutzer lesbar darzustellen. Er wird in Form eines einfachen Strings gespeichert.

2. Der **vorverarbeitete Text** ist das Ergebnis des Preprocessing (siehe Abschnitt 4.3). Er wird durch zwei Listen beschrieben, in denen jeweils das i-te Element mit dem i-ten Wort des Textes, dass nicht bei der Stoppwörter-Entfernung verloren ging, in Verbindung steht. Die eine Liste enthält die Übersetzung der Wörter in das Integer-Alphabet, wogegen die andere die Startpositionen der Wörter im Originaltext beschreibt.

3. Die **Term-Frequency-Vektor-Form** des Textes wird zusätzlich für die Verwendung im Candidate Retrieval verwendet. Aus Gründen der Modularität wurde diese nicht direkt in die Dokument-Datenstruktur aufgenommen, sondern im Retrieval-Index. So kann sichergestellt werden, dass `Document` immer noch aktuell bleibt, falls sich das Verfahren im Candidate Retrieval ändert.

Zusätzlich enthält die `Document`-Datenstruktur noch zusätzliche Meta-Informationen über das Dokument, wie z.B. seinen Titel oder die Namen seiner Autoren.

4.2.2 Umsetzung

Die Umsetzung einer Dokumentenbasis erfordert eine Datenstruktur, die die einzelnen `Document`-Objekte verwaltet und dabei die folgenden funktionalen Anforderungen erfüllt:

1. **Dokument hinzufügen** Jeden Tag werden Unmengen an neuen Texte verfasst, aus denen potenziell plagiiert werden kann. Daher muss es möglich sein, die Datenbasis um neue Dokumente zu erweitern. Hierbei muss jedoch ein Preprocessing unter

[1] `https://stats.wikimedia.org/EN/Sitemap.htm` (Zuletzt abgerufen: 06.04.2016)
[2] `https://www.wikipedia.org/` (Zuletzt abgerufen: 06.04.2016)
[3] `https://dumps.wikimedia.org/` (Zuletzt abgerufen: 06.04.2016)

Umständen neu durchgeführt werden. Das Löschen eines Dokuments aus der Basis hingegen ist ggf. deutlich unwichtiger und muss nicht zwingend unterstützt werden, wenn dies Performance-Gewinne mit sich bringt.

2. **Iteration über alle Dokumente** Im Candidate Retrieval müssen alle Dokumente der Dokumentenbasis mit einem verdächtigen Dokument verglichen werden. Dazu ist es notwendig, dass für alle Dokumente Term-Frequency-Vektoren gebildet werden (siehe Abschnitt 4.4). Diese Operation erfordert es, dass alle Dokumente der Dokumentenbasis nach und nach geladen werden und diese Berechnungen auf ihnen durchgeführt werden.

3. **Lesezugriff auf ein bestimmtes Dokument** Das Ergebnis des Candidate Retrieval ist eine Menge von Kandidatendokumenten, die eine (möglichst kleine) Teilmenge aller Dokumente in der Basis sind. Im Rahmen des Text Alignment muss nun auf diese einzelnen Dokumente zugegriffen werden.

Konzeptionell lassen sich diese Anforderungen gut mit einem einfachen Key-Value-Storage umsetzen. Als Schlüssel werden dabei fortlaufende Integer-Zahlen verwendet. Den zu einem Dokument gehörenden Schlüssel nennen wir *Document-ID*. Über diese kann auf einzelne Dokumente effizient zugegriffen werden. Die Iteration über alle Dokumente ist unabhängig von der Document-ID und ist ein einfacher linearer Scan über den Storage.

Zur Realisierung des Key-Value-Storage wird die SQL-Datenbank SQLite[4] verwendet. Alle Quelldokumente werden dafür als Records in einer Datenbank-Tabelle gespeichert. Als Primärschlüssel wird die Document-ID gewählt, da ausschließlich über diese auf Dokumente zugegriffen wird (wenn nicht ohnehin über die gesamte Datenbank iteriert wird). Die komplette Datenbank wird im Dateisystem in einer Datei `base.db` gespeichert und aus dieser geladen. Diese befindet sich in einem Verzeichnis mit dem Namen der Dokumentenbasis.

In der ersten Version von SciencePlag wurde die Dokumentenbasis noch durch eine Hashmap realisiert. Während für jedes Quelldokument ein `Document`-Objekt im Hauptspeicher lag, wurden dessen speicheraufwändigen Bestandteile, nämlich der Originaltext und der vorverarbeitete Text, in separaten Dateien im Dateisystem abgelegt. Der Vorteil einer vollständigen Datenbank gegenüber einzelner angelegter Dateien ist vielseitig:

1. Es wird ein Speicheroverhead durch einer Vielzahl von eventuell auch nur sehr kleinen Dateien vermieden. Die Blockgröße eines typischen Linux-Dateisystems (ext4) beträgt 4 Kilobyte. Das heißt, dass jede Datei immer das gemessen an ihrer Größe nächstgrößere Vielfache von 4 kB an Platz auf der Platte belegt. Beispielsweise ist die durchschnittliche Größe eines Wikipedia-Artikels 8.4 kB. Wenn jeder Artikel im Mittel 2 kB mehr Platz benötigt als notwendig, summieren sich dieser Speicheroverhead. SQLite speichert alle Daten in der einzelnen Datei `base.db` ab. Außerdem schont die geringere Zahl Dateizugriffe die Festplatte bzw. SSD des Nutzers.

2. Die Zugriffszeiten auf die einzelnen Dokumente, sind schneller, denn es werden weniger Systemaufrufe benötigt und der Zugriff erfolgt ausschließlich über den Primärschlüssel Document-ID.

3. SQL-Anfragen werden atomar behandelt und so kann das Programm, wird es abrupt abgebrochen, die Dokumentenbasis nicht in einem korrupten Zustand hinterlassen.

[4] `https://www.sqlite.org/` (Zuletzt abgerufen: 06.04.2016)

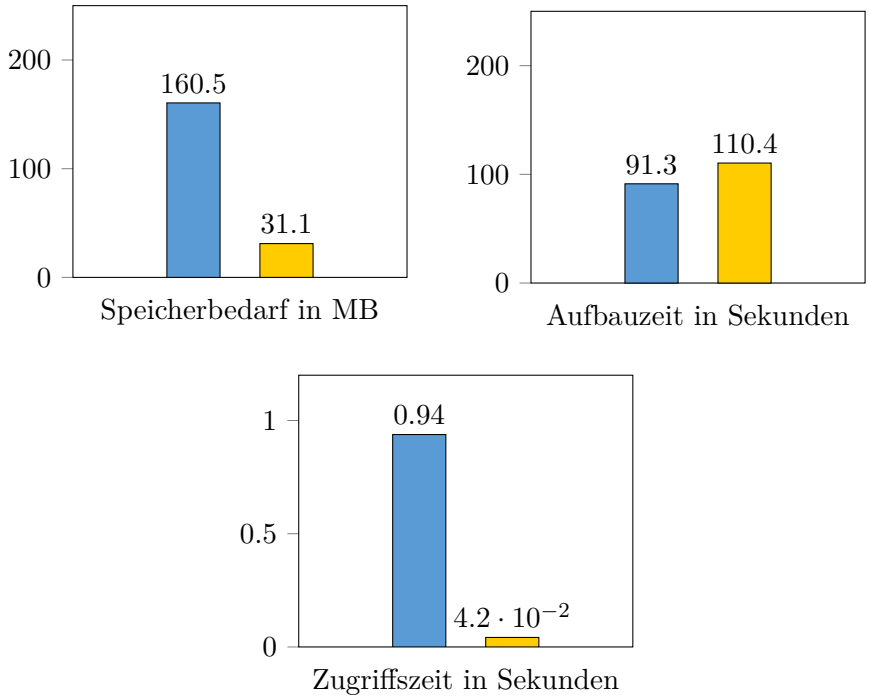


Abbildung 4.3: Vergleich des benötigten Speicherplatzes, der benötigten Zeit zum Aufbau einer Dokumentenbasis aus dem PAN-Korpus und Zeit für den Zugriff per Document-ID auf 1000 zufällig gewählte Dokumente (: Dateisystem, : SQLite)

In Abbildung 4.3 kann man erkennen, dass sich die Zeit zum Aufbau der Dokumentenbasis leicht verlängert hat, sie aber dafür deutlich weniger Speicherplatz benötigt. Auch die Zugriffszeiten bei Zugriffen via Document-ID (nicht abgebildet) sind signifikant verkürzt.

Der Zugriff auf die Dokumentenbasis erfolgt mit effizienten SQL-Anfragen. Diese ermöglichen unter anderem auch das blockweise Hinzufügen von Dateien.

4.3 Vorverarbeitung

Bereits in Abschnitt 2.3 wurde auf die typischen in der Vorverarbeitung notwendigen Schritte eingegangen. Darauf aufbauend soll an dieser Stelle weiter ins Detail und auf die Umsetzung der von uns gewählten Vorverarbeitungsschritte eingegangen werden.

Zunächst versuchten wir die Textvorverarbeitung in SciencePlag mit der Python-Bibliothek *Natural Language Toolkit*[5] (NLTK) umzusetzen. Diese stellt Algorithmen für sämtliche für die Plagiatssoftware benötigten Vorverarbeitungsschritte bereit. Dieses Vorgehen stellt sicher, dass wir zuverlässige, ausreichend getestete Verfahren nutzen, die zuvor bereits in zahlreichen Projekten genutzt wurden. Während es vorteilhaft ist für die gesamte Textvorverarbeitung nur eine einzige Bibliothek zu nutzen, so hat es sich als nicht zu vernachlässigender Flaschenhals herausgestellt, Python über seine C-Anbindung aus C++ heraus aufzurufen. Desweiteren bereitete die unterschiedliche Kodierung von Strings in den beiden Programmiersprachen Probleme, weshalb wir beschlossen haben, ein eigenes Vorverarbeitungsmodul zu implementieren.

Abbildung 4.4 liefert einen Überblick über die Preprocessing-Pipeline von SciencePlag. Das Preprocessing erhält als Eingabe ein Dokument in Form einer Zeichenkette.

[5] `http://www.nltk.org/` (Zuletzt abgerufen: 06.04.2016)

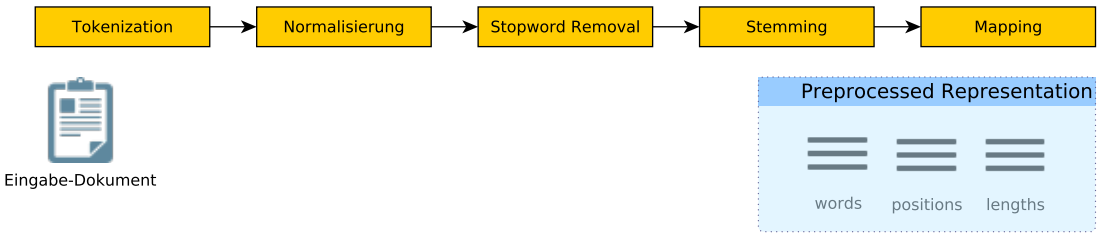


Abbildung 4.4: Überblick über die Preprocessing-Pipeline von SciencePlag

Am Ende der Vorverarbeitung wird das Dokument durch drei gleich lange Listen von nicht-negativen Integern repräsentiert:

`words`
: Der i-te Eintrag dieser Liste ist eine *Integer-Repräsentation* (s.u.) des i-ten (nicht entfernten) Token des Textes.

`positions`
: Der i-te Eintrag dieser Liste beschreibt die Anfangsposition des i-ten (nicht entfernten) Token im ursprünglichen Text, also an welcher Byteposition sich das erste Zeichen des Token befindet.

`lengths`
: Der i-te Eintrag dieser Liste ist die Anzahl von Bytes, die das i-te (nicht entfernte) Token im ursprünglichen Text besitzt.

Neben den bereits in Abschnitt 2.3 vorgestellten Schritten Segmentierung, Stammformreduktion und Stoppwörter-Entfernung, werden die Wörter nach der Zerlegung des Textes in Token noch einem Normalisierungsprozess unterworfen. Als letzter Schritt der Vorverarbeitung werden die Wörter auf Integer abgebildet, d.h. die Stringrepräsentationen eines Wortes wird durch eine eindeutige positive Zahl ersetzt (*Integer-Repräsentation*).

4.3.1 Tokenization

SciencePlag beschränkt sich bei der Tokenization auf einen sehr einfachen Ansatz: wir segmentieren den Text überall dort, wo er eine Folge von einem oder mehreren *Trennzeichen* enthält. Die Trennzeichen sind jede Form von Whitespace (z.B. Leerzeichen, Tabulatoren und Zeilenumbrüche), sowie die Symbole

. ! ? , : ; < > ( ) [ ] \ / “ ’ ‘ |

Wie in den Grundlagen beschrieben, hat dieser Ansatz einige Nachteile. Da unser Prozess der Plagiatserkennung jedoch ohnehin nicht das Finden von Verschleierungen mittels Synonymen vorsieht, vermuten wir, dass diese nicht signifikant sind. Die Vorteile dieses Vorgehens sind seine gute Performance (es ist nur ein Lauf über den Text notwendig), sowie seine Unabhängigkeit von der Textsprache und aufwändigen Informationsextraktionsverfahren, die domänen- und sprachabhängige annotierte Trainingsdatensätze benötigen.

Die aufgeführte Liste von Trennzeichen ist nicht besonders lang und erscheint unvollständig. Beispielsweise enthält sie keine französischen Anführungszeichen, die zweifelsohne auch als Trennzeichen wahrgenommen werden sollten. Der Grund dafür ist, dass Strings in C++ lediglich Folgen von Bytes sind und keine explizite Zeichenkodierung vorausgesetzt wird. Da in den meisten sinnvollen Kodierungen die Teilmenge der Symbole, die dem

ASCII-Datensatz entsprechen, auf die gleichen Zahlen abgebildet werden, erlauben wir als Trennzeichen nur ASCII-Symbole.

Bei der Tokenization reicht es nicht aus, einfach eine Liste von Wörtern auszugeben. Zur Visualisierung von plagiierten Textstellen benötigen wir ihre Anfangs- und Endposition im Originaltext. Daher speichert der Algorithmus zusätzlich die Anfangsposition und Länge eines jeden Token von diesem in den Listen `positions` und `lengths`.

4.3.2 Normalisierung

Um die Behandlung von Sonderzeichen in den nächsten Schritten zu vereinfachen, werden sie durch den Normalisierungsschritt einfach entfernt. Dabei dürfen nur so viele Symbole entfernt werden, dass keine Information verloren geht und die folgenden Stammformreduktions- und Stoppwörterentfernungsschritte noch vernünftig auf den Token arbeiten können. Da SciencePlag ausschließlich die Sprachen Deutsch und Englisch unterstützt, werden ausschließlich die ASCII-kodierten alphanumerischen Zeichen beibehalten und eventuell in Kleinschreibung überführt (d.h. in `a-z` und `0-9`), sowie die deutschen Umlaute und der Bindestrich. Das deutsche `ß` wird in `ss` umgewandelt.

Es kann passieren, dass bei der Normalisierung Token vollständig wegfallen, da alle ihre Zeichen entfernt werden. Daher werden ggf. auch die Listen `positions` und `lengths` noch verändert, indem Elemente aus ihnen gelöscht werden.

4.3.3 Stammformreduktion

Im Bezug auf SciencePlag kommt der Stammformreduktion eine besondere Bedeutung zu. Bei der bewussten Plagiierung von Dokumenten ist zu erwarten, dass nicht nur die Reihenfolge von Sätzen, sondern auch die Reihenfolge innerhalb der Sätze abgewandelt wird. Damit einher gehen oft veränderte grammatikalische Formen von Wörtern, die durch Stammformreduktion normalisiert werden können. Der Stammformreduktionsschritt ist der sprachabhängigste Bestandteil des gesamten Plagiatsfindungsprozesses. SciencePlag beinhaltet Stemmer für die deutsche und englische Sprache, bei denen es sich um Implementierungen des Snowball-Stemming-Algorithmus [16] handelt, wie sie auch im NLTK zum Einsatz kommen. Sie wurden allerdings in Details an unseren Zeichensatz und unsere Anforderungen angepasst. Die Stammformreduktion ist der aufwändigste Bestandteil des Vorverarbeitungsschritts, da es zahlreiche Läufe über die Token benötigt und sehr viele Fallunterscheidungen enthält.

4.3.4 Integer-Repräsentation

Einzelne Worte werden nicht als String, sondern als Integer vorgehalten. Diese Integer-Repräsentation ermöglicht das effiziente Vergleichen von Wörtern bei einer geringeren Speicherauslastung [4]. Dazu wird für jede Dokumentenbasis eine entsprechende Abbildung $\phi : \mathcal{W} \to \mathbb{N}$ erstellt, mit $\mathcal{W}$ als der Menge aller Worte, die in den vorverarbeiteten Dokumenten der Dokumentenbasis insgesamt vorkommen. Die Abbildung wird als Teil der Dokumentenbasis gespeichert. Dies ist u. a. nötig, um den Text eines verdächtigen Dokumentes für das Text Alignment in die Repräsentation der Dokumentenbasis zu überführen. Jedem neu auftretenden Wort wird die nächste freie Zahl zugewiesen. Das erste Wort wird auf 1 abgebildet, das zweite auf 2 und so weiter. Somit ergibt sich das *Integer-Alphabet* $I = \{1, \ldots, |\mathcal{W}|\}$.

4.4 Candidate Retrieval

Im Kontext der Projektgruppe verwenden wir Techniken aus dem Bereich Information Retrieval, um solche Dokumente zu finden, die eine starke Ähnlichkeit zu einem gegebenen verdächtigen Dokument besitzen. Ziel ist die Selektion von Dokumenten, die als Quellen für mögliche Plagiatsstellen des verdächtigen Dokuments in Frage kommen (siehe auch Abschnitt 2.4). Die Suche nach potentiell ähnlichen Dokumenten beschränkt sich hierbei auf die Dokumentenbasis. Das Ergebnis unseres IR-Ansatzes ist eine Filterung der Dokumentenbasis. Quelldokumente, die dem verdächtigen Dokument ähnlich sind, werden in der Menge von Kandidatendokumenten gesammelt. Die Ähnlichkeit zweier Dokumente basiert dabei auf dem in ihnen enthaltenen Wortschatz, wobei solchen Wörtern eine höhere Bedeutung für die (Un-)Ähnlichkeit zugewiesen wird, die in der Dokumentenbasis selten vorkommen. Hierzu verwenden wir das zuvor vorgestellte Kosinus-Ähnlichkeitsmaß basierend auf tf-idf-Vektoren. Die Kandidatendokumente werden später in einem weiteren Analyseschritt genauer mit dem verdächtigen Dokument verglichen, um Plagiatsstellen zu ermitteln (Abschnitt 4.5).

Voraussetzung für das Candidate Retrieval ist eine Menge bekannter Dokumente, die bereits als vorverarbeitete Listen von Wörtern existieren (Abschnitt 4.3). Zur weiteren Verwendung im Candidate Retrieval werden diese Dokumente in Termfrequenz-Vektoren überführt. Das Filtern ähnlicher Dokumente geschieht unter Angabe von einem von zwei möglichen Parametern. Zum einen kann bei Wahl von k als Anzahl der Ergebnisse direkt bestimmt werden, wie viele der ähnlichsten Dokumente ausgegeben werden sollen (top-k-Retrieval). Zum anderen kann über die minimale Ähnlichkeit $s \in [0, 1] \subset \mathbb{R}$ angegeben werden, dass sämtliche Dokumente mit einer gewissen Mindestähnlichkeit zum verdächtigen Dokument ermittelt werden sollen.

Zur Suche nach ähnlichen Dokumenten verwendet das Candidate Retrieval eine Indexstruktur, die in Abschnitt 4.4.1 beschrieben ist. Der eigentliche Suchvorgang wird in Abschnitt 4.4.2 näher erläutert, wobei von den einschränkenden Parametern k beziehungsweise s abstrahiert wird.

4.4.1 Retrieval Index

Das Candidate Retrieval verwendet eine Indexstruktur über alle aus Dokumenten bekannten vorverarbeiteten Wörter. Wird ein neues Dokument hinzugefügt, so kann die Indexstruktur durch einmaliges Iterieren über die Wörter im neuen Dokument $d = (t_1, t_2, ..., t_n)$ aktualisiert werden. Zusätzlich zur Indexstruktur selbst wird eine Liste der in ihr verwalteten Dokumente gespeichert.

Algorithmus 1 *AddDocument* beschreibt das Hinzufügen eines neuen Dokuments. Zu Beginn wird geprüft, ob das Dokument d bereits in die Indexstruktur aufgenommen wurde. In diesem Fall terminiert der Algorithmus frühzeitig, ohne die Indexstruktur zu verändern (Zeilen 1–2). Andernfalls wird das neue Dokument als Teil des Index registriert (Zeile 3). Der Index S selbst stellt eine zweidimensionale Datenstruktur dar. Die erste Dimension enthält sämtliche verwendeten Terme aus hinzugefügten Dokumenten. Für jeden Term t gibt es in der zweiten Dimension Einträge für diejenigen Dokumente d, in denen t vorkommt. Ein Eintrag ist ein Zähler, der die Termfrequenz von t in d wiedergibt. Wird ein neues, noch nicht im Index aufgenommenes Dokument hinzugefügt, so werden die Zähler entsprechend inkrementiert (Zeilen 4–5). Zuvor nicht vorhandene Einträge stattdessen mit 1 initialisiert.

Wie bei der Dokumentenbasis selbst (vgl. Abschnitt 4.2) ist Effizienz beim Retrieval Index aufgrund der großen Anzahl verwalteter Dokumente von Bedeutung, sowohl beim

AddDocument
Eingabe:
neues Dokument $d = (id, [t_1, t_2, ..., t_n])$ (ID mit Vektor aus Wörtern)
Indexstruktur $S[<Term>][<Dokument\text{-}ID>]$ aus Zählern
Liste D von Dokumenten in der Datenbasis
Ausgabe: Aktualisierter Index: S_{Res}
1: **if** Dokument mit ID id bereits in D **then return**
2: füge Dokument d zu D hinzu
3: **for** $i = 1 \dots n$ **do**
4: $S[t_i][id] \mathrel{+}= 1$

Algorithmus 1: *AddDocument*: Fügt ein Dokument (Liste von Wörtern) zur Indexstruktur S hinzu.

Speicherbedarf als auch bei der Zugriffszeit. Unsere Implementierung versucht, einen guten Kompromiss zwischen den beiden gegenläufigen Zielen Speicher- und Laufzeiteffizienz zu ermöglichen.

Die Indexstruktur wird von uns als Menge von *Posting Lists* implementiert. Für jedes im Korpus vorkommende Token existiert eine Posting List. Eine Posting List ist eine Liste, die alle Dokumente enthält, in der das entsprechende Token vorkommt, sowie die Häufigkeit des Tokens in diesem Dokument. Für die Darstellung der Menge verwenden wir eine Hashmap, die Terme auf Posting Lists abbildet. Grund hierfür ist die unter Umständen große, im Index repräsentierte Dokumentenbasis. Existiert ein Term nicht im verdächtigen Dokument, so wird für diesen Term kein Vergleich mit den Dokumenten im Index benötigt. Es müssen bei einem Aufruf des Candidate Retrieval also keinesfalls alle Terme im Index betrachtet werden, weshalb wir die Nutzung einer Hashmap einer Liste vorziehen. Wenn der benötige Speicherplatz der Dokumentenbasis in fortlaufenden Anwendungen stark anwächst, so könnten anstatt der Hashmap sortierte Listen bzw. Baumstrukturen verwendet werden (vgl. auch Abb. 4.5).

Ein Element einer Posting List zum Term t besteht aus einem Verweis auf ein Dokument d sowie der Häufigkeit von t in d. Die zweite Dimension der Indexstruktur wird also als Liste verwaltet. Motivation hierfür ist die Berechnung der Kosinus-Ähnlichkeit des verdächtigen Dokuments mit den Dokumenten im Index (vgl. Abschnitt 4.4.2). Da unsere Auswertung ein nicht-approximatives Ranking der Indexdokumente vornimmt, muss in der Regel für alle in einer betrachteten Posting List vorkommenden Dokumente ein Ähnlichkeitswert berechnet beziehungsweise aktualisiert werden.

Die Speicherung des Retrieval-Index findet mit Hilfe der Serialisierungs-Bibliothek *Cereal*[6] statt. Es handelt sich dabei um eine Header-Only-Bibliothek, die einfach in das Projekt einzubinden und zu benutzen ist. Zusätzlich unterstützt sie die Speicherung von Containern der C++-Standardbibliothek ohne zusätzlichen Implementierungsaufwand. In einem Benchmarkvergleich[7] verschiedener Serialisierungsverfahren zeigt sich, dass Cereal im Bezug auf die zur Serialisierung- und Deserialisierung benötigte Zeit sehr gut geeignet ist. Cereal unterstützt sowohl binäre Serialisierung, als auch Serialisierung nach XML. Der Vorteil des XML-Formates ist, dass die Inhalte der Dateien menschenlesbar bleiben und somit das Debuggen leichter fällt. In späteren Versionen des Programms verwendeten wir die binäre Serialisierung wegen des niedrigeren Speicherbedarfs und wegen der höheren Geschwindigkeit.

[6] `http://uscilab.github.io/cereal/` (Zuletzt abgerufen: 06.04.2016)
[7] `https://github.com/thekvs/cpp-serializers` (Zuletzt abgerufen: 06.04.2016)

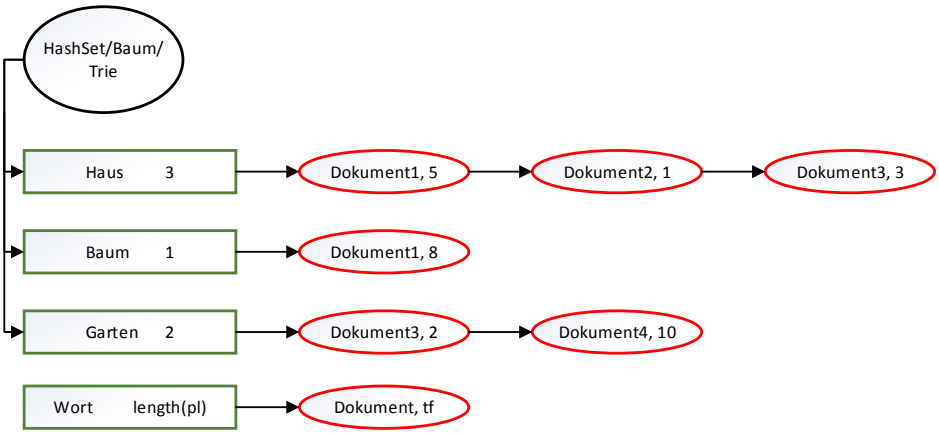


Abbildung 4.5: Posting Lists für die Wörter *Haus*, *Baum* und *Garten*, die in Dokumenten 1 bis 4 vorkommen. Eine Posting List enthält das jeweilige Wort, die Länge dieser Posting List sowie Listenelemente. Die Elemente beschreiben, in welchen Dokumenten das Wort wie häufig enthalten ist.

4.4.2 Kosinus-Ähnlichkeit

Zur Ermittlung der Kosinus-Ähnlichkeit zwischen dem verdächtigen Dokument und den Elementen der Dokumentenbasis wird die zuvor erwähnte Indexstruktur in Form von Posting Lists verwendet. Die Berechnung erfolgt gemäß Algorithmus 2 (*CosineScore*). Als Eingabe erwartet der Algorithmus das verdächtige Dokument als Termfrequenz-Vektor, der als Liste von Paaren (<Term>, <Termfrequenz>) dargestellt ist. Weitere Eingaben sind die Indexstruktur als Menge von Posting Lists sowie eine erweiterte Version der Dokumentenbasis, die bereits Termfrequenz-Vektoren für jedes Dokument beinhaltet. Die Ausgabe ist eine Abbildung von den Elemente der Dokumentenbasis auf ihre Ähnlichkeitswerte beim Vergleich mit der Anfrage. Jeder Eintrag dieses Vektors drückt die Kosinus-Ähnlichkeit der tf-idf-Vektoren des verdächtigen Dokuments und eines Dokuments der Dokumentenbasis aus. Die Ausgabe enthält genau einen Eintrag pro Element der Dokumentenbasis.

Zu Beginn von CosineScore wird der Ähnlichkeitswert zur Anfrage für jedes Element der Dokumentenbasis zu Null initialisiert (Zeile 1). Desweiteren wird für jedes Dokument inklusive der Anfrage die euklidische Norm des jeweiligen Termfrequenz-Vektors berechnet. Bei mehreren Aufrufen von Algorithmus 2 muss diese Berechnung für Elemente der Dokumentenbasis nur einmal durchgeführt werden.

Nach diesen Vorbereitungen iteriert der Algorithmus über die Terme des verdächtigen Dokuments (Zeile 3). Für jeden solchen Term t wird die entsprechende Posting List aus der Indexstruktur und die Länge dieser Posting List abgerufen (Zeilen 4–5). Diese Länge entspricht der Anzahl Dokumente in der Dokumentenbasis, in denen dieser Term vorkommt, der so genannten *Document Frequency*. Im Anschluss wird der tf-idf-Wert von t innerhalb der Anfrage berechnet (Zeile 6, vgl. auch Definition 2.1). Die Funktion max-tf(q) berechnet die maximale Termfrequenz, die in der Anfrage vorkommt. Für jedes Dokument d in der zum Term t gehörenden Posting List wird der tf-idf-Wert von t in d berechnet und der Ähnlichkeits-Score aktualisiert (Zeilen 7–8). Bevor der Vektor der Ähnlichkeitswerte zurückgegeben wird, werden sämtliche Einträge normalisiert, um eine korrekte Berechnung der Kosinus-Ähnlichkeit zu gewährleisten (Zeilen 9–10, vgl. Satz 2.2).

CosineScore
Eingabe:
Anfrage $q = [(t_1, f_1), (t_2, f_2), ..., (t_n, f_n)]$
Indexstruktur S (Menge von Posting Lists)
Dokumentenbasis D mit Termfrequenzen (Elemente haben die gleiche Struktur wie q)
Ausgabe: Ähnlichkeiten *Scores* im Bereich $[0, 1]$ von Dokumenten aus D zu q

initialisiere *Scores* für alle $d \in D$ mit 0
initialisiere M für alle $d \in (\{q\} \cup D)$ mit $\|\text{tf-idf-vector}(d)\|_2$
for all Termfrequenzpaar $(t, f) \in q$ **do**
    $p \leftarrow$ Posting List aus S zu Term t
    $df \leftarrow \text{length}(p)$
    $\alpha \leftarrow \text{tf-idf-weight}(f, \text{max-tf}(q), df, |D|)$
    for all $(d, f_{t,d}) \in p$ **do**
        $Scores[d] \mathrel{+}= \alpha \cdot \text{tf-idf-weight}(f_{t,d}, \text{max-tf}(d), df, |D|)$
for all Dokument $d \in D$ **do**
    $Scores[d] \leftarrow Scores[d]/(M[q] \cdot M[d])$
return *Scores*

Algorithmus 2: Algorithmus *CosineScore*: Berechnet die Kosinus-Ähnlichkeiten des verdächtigen Dokuments q mit sämtlichen Elementen der Dokumentenbasis D.

4.5 Text Alignment

Das *Text Alignment* bezeichnet die eigentliche Suche nach Plagiatsstellen und erfolgt nach dem Candidate Retrieval. Das verdächtigte Dokument wird zusammen mit einer Menge von Kandidatendokumenten analysiert, um mögliche Plagiate zu finden. Resultat des Text Alignment ist eine Menge von Fundstellen. Dabei ist eine Fundstelle ein Textabschnitt im verdächtigen Dokument, dem eine Textstelle in einem der Kandidatendokumenten zugeordnet wird, d.h. ein Textabschnitt, von dem vermutet wird, dass dieser plagiiert wurde. Die Text-Alignment-Komponente arbeitet auf dem Integer-Alphabet, welches durch die Candidate Retrieval Komponente erstellt wird, wobei die gewählte Zuordnung von Wörtern auf Zahlen hier nicht relevant ist. Das Text Alignment erfolgt durch die folgenden Schritte:

1. Aufbau eines Suffixarrays über dem verdächtigem Dokument und den Kandidatendokumenten.
2. Finden aller gemeinsamer Teilstrings zwischen verdächtigem Dokument und Kandidatendokument.
3. Postprocessing: Filterung und Vereinigung von Fundstellen.

Im Folgenden werden die einzelnen Schritte im Detail beschrieben: Abschnitt 4.5.1 beschreibt die ersten beiden Schritte, das Postprocessing wird in Abschnitt 4.5.2 beschrieben. Folgende Symbole werden genutzt, um Alignment und Postprocessing genauer zu beschreiben:

Definition (Fundstelle). Sei $f = ((s_\text{v}, l_\text{v}), (s_\text{k}, l_\text{k}), id)$ eine Fundstelle mit folgenden Bezeichnungen:

- s_v: die Startposition im verdächtigen Dokument,
- l_v: die Länge der Textstelle im verdächtigen Dokument,

- s_k: die Startposition im Kandidatendokument,
- l_k: die Länge der Textstelle im Kandidatendokument,
- *id*: die ID des Kandidatendokuments, in welchem sich die Fundstelle befindet.

Der Vorteil, die Länge der Textstelle im verdächtigen Dokument und Kandidatendokument getrennt zu betrachten, ist in den späteren Alignment-Schritten zu sehen. Zunächst unterscheiden sich die Werte beider Stellen nicht. Stellen können aber noch mit anderen überlappenden Stellen so vereinigt werden, dass kleinere eingeschobene Lücken entstehen, sodass sich die Längen am Ende unterscheiden. Eine Fundstelle beschreibt damit eine *verdächtige Stelle* im Kontext des Text Alignment. Um nicht auch beispielsweise einzelne Wörter als eine solche Fundstelle auszugeben, wird der folgende Schwellwert $MinLCP$ genutzt.

Definition (MinLCP). Der Wert $MinLCP \in \mathbb{N}_+$ bestimmt die minimale Länge eines möglichen Plagiats. Das heißt eine Textstelle muss sich aus mindestens $MinLCP$ Wörtern zusammensetzen, um als mögliches Plagiat in Betracht gezogen zu werden.

4.5.1 Alignment

Als Eingabe bekommt das Text Alignment das verdächtige Dokument v, die n passenden Kandidatendokumente $K = \{k_1, \ldots, k_n\}$ (in Integer-Repräsentation) sowie den Schwellwert $MinLCP$. Alle Eingabedokumente sind vorverarbeitet und können direkt zum Aufbau des Suffixarrays genutzt werden. Dazu werden sie zunächst zu einem Text T konkateniert, wobei das verdächtige Dokument an den Anfang gesetzt wird, d.h. $T := [v[1], \ldots v[|v|], k_1[1], \ldots, k_1[|k_1|], \ldots, k_n[1], \ldots, k_n[|k_n|]]$[8]. Sei im Folgenden $T[i]$ das i-te Wort in T. Die Startpositionen der jeweiligen Dokumente werden im Hilfsarray d gespeichert. Es gilt $T[d(i)] = k_i[1]$ und $T[d(0)] = v[1]$. Das Suffixarray und LCP-Array werden nun über T aufgebaut. Für $T[i]$ sei $D[i] \in \{0, \ldots, n\}$ der Index des Dokuments, aus welchem das Wort $T[i]$ entstammt, daher $1 \leq i \leq |v| : D[i] = 0$ und $|v| + \sum_{j=1}^{m-1} |k_j| \leq i \leq |v| + \sum_{j=1}^{m} |k_j| : D[i] = m$.

Im Folgenden werden Suffixarray und LCP genutzt, um Textstellen zu finden, welche sowohl im verdächtigen Dokument, sowie in mindestens einem Kandidatendokument vorkommen. Dazu werden Suffixarray und LCP-Array vorwärts durchlaufen und jeder Eintrag SA[i] (entspricht Verweis auf eine Position in T) mit dem Hilfsarray d geprüft, aus welchem Dokument das Wort T[SA[i]] stammt. Es kann mit einem Vergleich festgestellt werden, ob der Suffixarray-Eintrag auf das verdächtige Dokument verweist. In diesem Fall wird zunächst geprüft, ob die Distanz von T[SA[i]] bis zum Beginn des ersten Kandidatendokuments den Mindestwert $MinLCP$ erreicht (denn der Index des Suffixarrays könnte auf eines der letzten Wörter des ersten Dokumentes zeigen) und falls dies der Fall ist, ob mindestens eine der folgenden beiden Bedingungen zutrifft:

1. Der LCP-Wert des aktuell betrachteten Eintrags i ist größer als der Schwellwert $MinLCP$.
2. Der nachfolgende Suffixarray-Eintrag hat einen LCP-Wert größer als der Schwellwert $MinLCP$.

[8] Hier wäre eine Trennung der einzelnen Dokumente mit speziellen Trennzeichen $\$_1, \ldots, \$_n$ möglich, um ein Überlesen über Dokumentgrenzen zu verhindern. Dies ist in unserer Umsetzung jedoch nicht benötigt, da wir die entsprechenden Fälle explizit abfangen (s. u.).

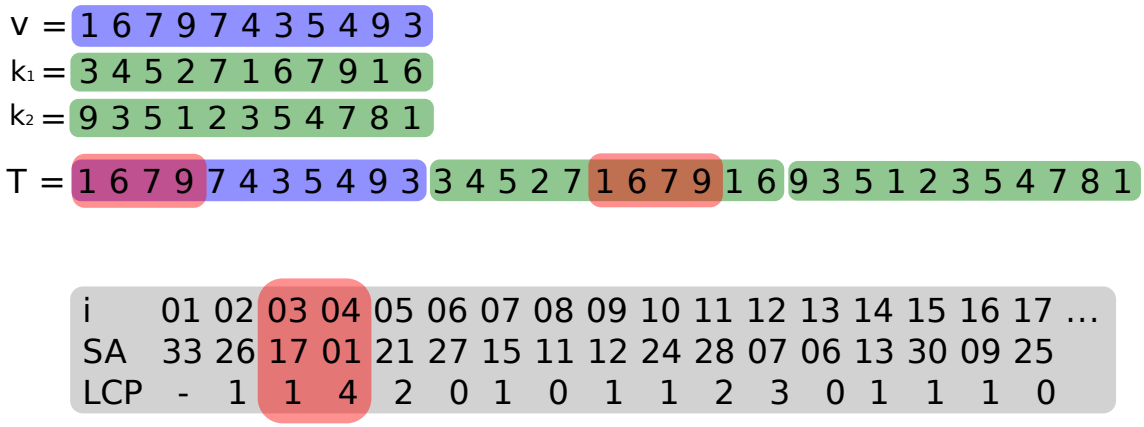


Abbildung 4.6: Zunächst werden die Eingabedaten, das verdächtiges Dokument v und die Kandidatendokumente k_1 und k_2, konkateniert. Anschließend werden Suffixarray und LCP aufgebaut. Gemeinsame Textstellen ausreichender Länge, werden anhand des Suffixarray- und LCP-Eintrags identifiziert.

Fall 1: Da der LCP-Wert des aktuellen Suffixarray-Eintrags SA$[i]$ größer als der gegebenen Schwellwert $MinLCP$ ist, existiert der direkte Vorgängereintrag SA$[i-1]$. Diese beiden Einträge teilen sich einen gemeinsamen Präfix der Länge LCP$[i]$. Gilt nun, dass SA$[i-1]$ auf ein Kandidatendokument verweist und diese Position mindestens $MinLCP$ von der Position e des letzten Worts des Dokuments in T entfernt ist, wurde eine Textstelle der Länge $l = \min\{\text{LCP}[i],\ e - \text{SA}[i-1]\}$ gefunden, welche in dem verdächtigem Dokument und dem Kandidatendokument vorkommt (siehe Abbildung 4.6). Die Fundstelle lautet damit $f = ((\text{SA}[i],\ l),\ (\text{SA}[i-1],\ l),\ D[i-1])$.

Falls LCP$[i-1] \geq MinLCP$ kann die Suche weiter fortgesetzt werden, das heißt, dass nun geprüft wird, ob diese Bedingungen für SA$[i-1]$ gelten. Wobei die Länge l_w einer weiteren gefundenen Textstelle, nicht die ursprüngliche Länge l überschreiten kann. Diese Rückwärtssuche in Suffixarray und LCP-Array kann dann fortgesetzt werden, das heißt als nächstes wird LCP$[i-2]$ betrachtet und so weiter. Die Suche endet, wenn ein Index k erreicht wird, für den LCP$[k]$ nicht mehr den Schwellwert überschreitet.

Fall 2: Betrachten wir den aktuellen Suffixarray-Eintrag SA$[i]$ und hat dessen direkter Nachfolgeeintrag SA$[i+1]$ einen LCP-Wert größer $MinLCP$, dann existiert ein gemeinsamer Präfix der Länge LCP$[i+1]$ für die beiden Suffixe SA$[i]$ und SA$[i+1]$. Befindet sich der Präfix von SA$[i+1]$ in einem der Kandidatendokumenten und an einer Stelle weit genug von dessen Ende entfernt, wurde eine gemeinsame Textstelle gefunden, d.h. die Fundstelle $f = ((\text{SA}[i],\ l),\ (\text{SA}[i+1],\ l),\ D[i+1])$ mit $l = \min\{\text{LCP}[i+1],\ e - \text{SA}[i+1]\}$ und e Position des letzten Wortes vom betrachteten Kandidatendokument in T. Die Suche kann auch hier weiter fortgesetzt werden, d.h. betrachte SA$[i+2]$ bzw. LCP$[i+2]$ und so weiter, bis ein Index k erreicht wird, für den LCP$[k]$ den Schwellwert $MinLCP$ nicht überschreitet.

Suffix	LCP	Dok.-ID	Index	Kommentar
0 4 5 7 ...	-	1	i-3	Keine Fundstelle: $LCP[i-2] < MinLCP$
5 0 1 5 ...	0	2	i-2	Fall 1
5 0 1 8 ...	2	3	i-1	Fall 1
5 0 2 7 ...	2	0	i	Aktuell betrachtet
5 0 3 4 ...	2	3	i+1	Fall 2
5 0 4 3 ...	2	4	i+2	Fall 2
5 0 4 7 ...	3	0	i+3	Keine Fundstelle, wegen Dok.-ID
5 1 3 0 ...	1	3	i+4	Keine Fundstelle: $LCP[i+4] < MinLCP$

Fall 1

Fall 2

Abbildung 4.7: Beispiel zur Fallunterscheidung mit $MinLCP = 2$: Aktuell wird das Suffix *5 0 2 7 ...* betrachtet. Von dort wird jeweils rückwärts und vorwärts im Suffixarray nach dem gleichen Präfix gesucht. Diese Suche erfolgt durch die LCP-Werte des aktuellen Eintrags $i - x$ mit $x \geq 0$ (Fall 1), sowie des Nachfolgeeintrags $i + y$ mit $y \geq 1$ (Fall 2) durchgeführt. So werden vier Fundstellen mit dem gemeinsamen Präfix *5 0* der Länge 2 gefunden.

FindPassages

Eingabe:
verdächtiges Dokument v
Menge der Kandidatendokumente $K = \{k_1, \ldots, k_j\}$
Minimallänge des gemeinsamen Präfix $MinLCP$

Ausgabe: Menge von Fundstellen F

1: Konkateniere Text aus v und Texte in K zu Text T
2: erstelle Suffixarray SA zu T
3: Fundstellen $F \leftarrow \emptyset$
4: **for** $i = 1..|T|$ **do**
5: **if** SA[i] $\leq$ Länge des verdächtigen Dokuments **then**
6: $F \leftarrow F \cup$ {Fundstellen für SA[i] gemäß Fall 1}
7: $F \leftarrow F \cup$ {Fundstellen für SA[i] gemäß Fall 2}
8: **return** F

Algorithmus 3: *FindPassages:* Findet verdächtige Stellen.

Algorithmus 3 *FindPassages* beschreibt das Vorgehen beim Text Alignment mit Hilfe der beschriebenen Fälle und Abbildung 4.7 zeigt ein Beispiel für das Vorgehen beim Alignment, wobei beide Fälle behandelt werden. Man sieht wie ausgehend von der aktuellen Position i die Rückwärts- und die Vorwärtssuche nach Fundstellen durchgeführt wird.

Bis jetzt beziehen sich die Fundstelle noch auf Indizes des Suffixarrays und müssen derart angepasst werden, dass sie sich auf Indizes innerhalb der Dokumente beziehen. Ist ein Wort $T[i]$ gegeben und die Position p des Wortes im Kandidatendokument $m = D[i]$ zu bestimmen, muss zunächst die Gesamtlänge der Dokumente, welche vor dem m-ten Dokument in T liegen bestimmt werden. Die Position ergibt sich als $p = i - (|v| + \sum_{j=1}^{m-1} |k_j|)$. Diese Anpassung wird während des Ablaufs von FindPassages durchgeführt, sobald eine neue Fundstelle aufgenommen wird.

Die ausgegebenen Fundstellen überschneiden sich teilweise. Denn sei eine Folge (z. B. ein Satz) von k Wörtern im verdächtigen Dokument gegeben, die auch in einem der Kandidatendokumente in gleicherweise vorkommt. Dann enthält das Suffixarray alle k möglichen Suffixe des Satzes: einmal für das verdächtige Dokument und einmal für das Kandidatendokument. Die Verweise auf das Suffix im verdächtigen Dokument und auf das Suffix im

Kandidatendokument, liegen jeweils als Nachbarn im Suffixarray vor (wegen dessen lexikographischem Aufbau). Daher wird eine gleiche Textstelle für alle $k - MinLCP$ Suffixe, welche den Schwellwert für die minimale Präfixlänge $MinLCP$ überschreiten gemeldet. Eine Aufgabe des Postprocessing ist das Verschmelzen dieser sich überlappenden Fundstellen zu einer gemeinsamen Fundstelle.

4.5.2 Postprocessing

Wie bereits in Kapitel 4.5.1 angemerkt ist Teil der Aufgabe des Postprocessing der Text Alignment Komponente das Verschmelzen von sich überlagernder Fundstellen. Dies ist Aufgabe der folgenden vier Teilschritte, welche das Gesamtergebnis verbessern sollen:

1. *Merge:* Der bereits erwähnte Schritt des Verschmelzens sich überlappender sowie nahe beieinander liegender Fundstellen.

2. *Delete:* In diesem Schritt werden alle Fundstellen entfernt, welche häufig vorkommenden Formulierungen entsprechen.

3. *Dominate:* Werden gleiche Fundstellen in verschiedenen Kandidaten gefunden, wird eine dieser ausgewählt und die übrigen entfernt.

4. *Filter:* Fundstellen, die eine Mindestlänge nicht erreichen werden verworfen.

Das Postprocessing reduziert somit die Anzahl der Fundstellen durch Verschmelzen, Löschen und Auswahl, so dass eine aussagekräftige Menge von möglichen Plagiaten übrig bleibt. Jedoch werden dabei Quellenangaben in Nähe der Fundstellen noch nicht gesucht und mit in Betracht gezogen, so dass Plagiate und korrekte Zitate bisweilen gleichermaßen gefunden und ausgegeben werden. Die einzelnen Postprocessingschritte werden im Folgenden detailliert beschrieben.

Merge

Das *Merging* ist der erste und wichtigste Schritt des Postprocessings. Es werden nicht nur alle sich überlagernden Fundstellen zusammengefasst, sondern auch alle diejenigen Fundstellen, welche sich in direkter Nähe zueinander befinden. Diese Nähe wird durch zwei Schwellwertparameter Δ_k und Δ_v angeben. Zwei Fundstellen werden genau dann miteinander verschmolzen, wenn folgende Bedingungen gelten:

1. Sie beziehen sich auf das gleiche Kandidatendokument d.

2. Der Abstand zwischen ihnen im Kandidatendokument ist kleiner als Δ_k.

3. Der Abstand zwischen ihnen im verdächtigen Dokument ist kleiner als Δ_v.

Seien nun die beiden Fundstellen $f = ((s_\mathrm{v},\ l_\mathrm{v}),\ (s_\mathrm{k},\ l_\mathrm{k}),\ d)$ und $g = ((s'_\mathrm{v},\ l'_\mathrm{v}),\ (s'_\mathrm{k},\ l'_\mathrm{k}),\ d)$ in einem Merge-Schritt gegeben. Der Index des letzten Wortes der gefundenen Textstelle bzgl. f ergibt sich als $e_\mathrm{v} := s_\mathrm{v} + l_\mathrm{v}$ im verdächtigen Dokument und $e_\mathrm{k} := s_\mathrm{k} + l_\mathrm{k}$ im Kandidatendokument, analog $e'_\mathrm{v} := s'_\mathrm{v} + l'_\mathrm{v}$ und $e'_\mathrm{k} := s'_\mathrm{k} + l'_\mathrm{k}$ für g. Gilt $s_\mathrm{v} \leq s'_\mathrm{v}$ dann sei der Abstand im verdächtigen Dokument definiert durch $d_\mathrm{v} := s'_\mathrm{v} - e_\mathrm{v}$ und der im Kandidatendokument durch $d_\mathrm{k} := s'_\mathrm{k} - e_\mathrm{k}$. Für Bedingungen 2 und 3 ergibt sich also $d_\mathrm{v} < \Delta_v$ und $d_\mathrm{k} < \Delta_\mathrm{k}$. Um sicherzustellen, dass $s_\mathrm{v} \leq s'_\mathrm{v}$ gilt, werden alle Fundstellen zunächst nach dem Startindex im verdächtigem Dokument sortiert und anschließend stabil

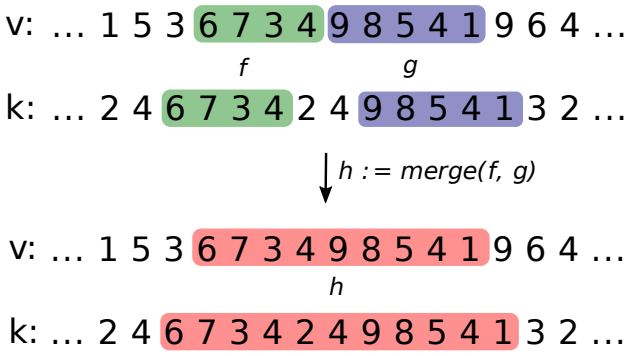


Abbildung 4.8: Erste Phase im Preprocessing: Das Verschmelzen sich überlappender oder nahe beieinander liegender Fundstellen. V ist das verdächtige Dokument, T ist das Kandidatendokument. Durch den Aufruf von Merge(f, g) werden die beiden Fundstellen f und g zu einer gemeinsamen Fundstelle h verschmolzen. Auslassungen oder Einfügungen von Wörtern können hierbei ausgeglichen werden.

nach Startindex im Kandidatendokument sowie Dokumenten-ID sortiert[9]. Damit wird zudem gewährleistet, dass aufeinander folgende sich überlappende Fundstellen der Reihe nach zu einer gemeinsamen Fundstelle zusammengefügt werden. Nach dem Sortieren reicht ein Durchlauf über alle Fundstellen eines Kandidatendokuments, um alle Merge-Schritte dafür durchzuführen. Das Resultat des Verschmelzens zweier Fundstellen f und g ist die neue gemeinsame Fundstelle $h = ((s''_\mathrm{v},\ l''_\mathrm{v}),\ (s''_\mathrm{k},\ l''_\mathrm{k}),\ d)$. Für diese gilt:

- $s''_\mathrm{v} := s_\mathrm{v}$ und $l''_\mathrm{v} := e'_\mathrm{v} - s_\mathrm{v} = s'_\mathrm{v} + l'_\mathrm{v} - s_\mathrm{v}$
- $s''_\mathrm{k} := s_\mathrm{k}$ und $l''_\mathrm{k} := e'_\mathrm{k} - s_\mathrm{k} = s'_\mathrm{k} + l'_\mathrm{k} - s_\mathrm{k}$

Durch die Wahl von Δ_v und Δ_k kann eine heuristische Suchqualität eingeführt werden. Abbildung 4.8 zeigt einen Merge-Schritt in dem die beiden Fundstellen f und g zu einer gemeinsamen Fundstelle h verschmolzen werden. Wie zu sehen liegen die Fundstellen im verdächtigen Dokument (V) direkt nebeneinander, wohingegen diese im Kandidatendokument (T) durch die beiden Worte 2 und 4 getrennt sind. Eine Wahl von $\Delta_v > 0$ und $\Delta_k > 0$ ermöglicht hier ein Finden von Plagiaten in denen Wörter zur Verschleierung entfernt oder auch eingefügt wurden. Dies funktioniert, solange in der Nachbarschaft ausreichend lange Teilstücke zu finden sind und Δ_v sowie Δ_k passend gewählt werden.

Der Algorithmus 4 *Merge* stellt dieses Vorgehen dar. Die dabei verwendete Subroutine *IsSuperResult* stellt fest, ob eine Fundstelle durch eine zweite vollkommen abgedeckt wird und die Subroutine vereinigt die Fundstellen wie beschrieben.

Delete

Der zweite Schritt des Postprocessing entfernt häufig vorkommende Formulierungen. Das bedeutet, eine Fundstelle, welche in genügend vielen Kandidatendokumenten ausreichend oft vorkommt, wird als unter Umständen domänenspezifische häufig genutzte Formulierung betrachtet und aus der Resultatmenge entfernt. Die maximale Häufigkeit einer Formulierung wird durch den Schwellwert *CommonMin* angegeben. Wird dieser überschritten, werden alle Fundstellen, welche sich auf diese Formulierung beziehen, entfernt. Um festzustellen, wie viele Fundstellen sich auf jeweils gleiche Textbereiche des verdächtigen Dokumentes beziehen, werden folgende Schritte angewandt, wobei die Fundstellen nach dem Index im verdächtigen Dokument sortiert vorliegen müssen:

[9] Stabil bedeutet hier, dass die Reihenfolge der Fundstellen mit gleichem Startindex bzw. Dokumenten-Id beibehalten wird.

Merge

Eingabe:
Menge von Fundstellen F
Minimalabstände Δ_v und Δ_k

Ausgabe: Menge von Fundstellen $F' \subseteq F$

1: Sortiere F nach Startindex im verdächtigem Dokument
2: Sortiere F stabil nach Startindex im Kandidatendokument
3: Sortiere F stabil nach Dokumenten-Id
4: Ergebnismenge $F' \leftarrow \emptyset$
5: $next \leftarrow True$
6: **for** $i = 1 \ldots |F|$ **do**
7: **if** $next$ **then**
8: $t \leftarrow F[i]$ ▷ Nächste Fundstelle laden
9: $next \leftarrow False$
10: **if** $i + 1 < |F|$ **and** $F[i+1].\mathrm{id} = t.\mathrm{id}$ **then**
11: $n \leftarrow F[i+1]$
12: $d_\mathrm{v} \leftarrow n.s_\mathrm{v} - (t.s_\mathrm{v} + t.l_\mathrm{v})$ ▷ Berechne Abstände
13: $d_\mathrm{k} \leftarrow n.s_\mathrm{k} - (t.s_\mathrm{k} + t.l_\mathrm{k})$
14: **if** $t.\mathrm{isSuperResult}(n)$ **or**
15: $(d_\mathrm{v} < \Delta_v$ **and** $abs(d_\mathrm{v}) < n.l_\mathrm{v} + \Delta_v$ **and**
16: $d_\mathrm{k} < \Delta_k$ **and** $abs(d_\mathrm{k}) < n.l_\mathrm{k} + \Delta_k)$ **then**
17: $t.\mathrm{merge}(n)$ ▷ Fundstellen werden vereinigt
18: **else**
19: $next \leftarrow True$ ▷ Keine weiteren Fundstellen nahe genug
20: **else**
21: $next \leftarrow True$ ▷ Merge für Dokumenten-Id abgeschlossen
22: **if** $next$ **then**
23: $F' \leftarrow F' \cup \{t\}$ ▷ Füge zusammengefügte Fundstelle zur Ergebnismenge hinzu
24: **return** F'

Algorithmus 4: *Merge*: Führt sich überlappernde oder nah beieinander liegende Fundstellen zusammen. Erster Schritt des Postprocessing.

1. Initialisiere ein $i := 0$.
2. Wähle i-te Fundstelle $f = ((s_\mathrm{v}, l_\mathrm{v}),\ (s_\mathrm{k}, l_\mathrm{k}),\ d)$.
3. Solange die nächste Fundstelle $g = ((s'_\mathrm{v}, l'_\mathrm{v}),\ (s'_\mathrm{k}, l'_\mathrm{k}),\ d')$ existiert und den gleichen Index und Länge bzgl. dem verdächtigen Dokument besitzt, d.h. $s_\mathrm{v} = s'_\mathrm{v}$ und $l_\mathrm{v} = l'_\mathrm{v}$, füge g einer temporären Liste L hinzu und inkrementiere i.
4. Falls Länge der Liste $L \leq$ *CommonMin*, füge diese und f der Ergebnismenge hinzu. Leere L und gehe zu 2 falls weitere Fundstelle existieren.

Dieser Algorithmus betrachtet jede Fundstelle einmal und entfernt genau diejenigen, welche sich zusammen hinreichend oft, d.h. mindestens *CommonMin* mal, auf eine Textstelle im verdächtigen Dokument beziehen und jeweils die gleiche Länge besitzen. Dies können beispielsweise domänenspezifische Formulierungen sein, die durch die Stammformreduktion bzw. die Stoppwortentfernung des Preprocessing nicht erfasst wurden.

Dominate

Der folgende Arbeitsschritt dient dazu, Ergebnisse mit redundanten Informationen herauszufiltern. Dies bedeutet, dass mehrere Ergebnisse, welche sich auf das gleiche Plagiat aus verschiedenen Quelle beziehen, zu einem Ergebnis zusammengefasst werden. Genauer werden im Folgenden alle Kombinationen von je zwei Fundstellen $f = ((s_\mathrm{v},\ l_\mathrm{v}),\ (s_\mathrm{k},\ l_\mathrm{k}),\ d)$ und $g = ((s'_\mathrm{v},\ l'_\mathrm{v}),\ (s'_\mathrm{k},\ l'_\mathrm{k}),\ d')$ untersucht. Unter folgenden Bedingungen wird der Algorithmus fortgesetzt, ansonsten wird mit der nächsten Fundstellen-Kombination fortgefahren:

1. $s'_\mathrm{v} \in [s_\mathrm{v}, s_\mathrm{v} + l_\mathrm{v})$
2. $(s'_\mathrm{v} + l'_\mathrm{v}) \in (s_\mathrm{v}, s_\mathrm{v} + l_\mathrm{v}]$
3. $s'_\mathrm{k} \in [s_\mathrm{k}, s_\mathrm{k} + l_\mathrm{k})$
4. $(s'_\mathrm{k} + l'_\mathrm{k}) \in (s_\mathrm{k}, s_\mathrm{k} + l_\mathrm{k}]$

In diesem Fall wird f als dominierende Fundstelle bezeichnet. Die Dokumente d und d' sind garantiert verschieden, da die beiden Fundstellen f und g ansonsten bereits im Arbeitsschritt *Merge* zu einem Ergebnis verschmolzen worden wären. Sind oben genannte Bedingungen erfüllt, so kann davon ausgegangen werden, dass f einen höheren Informationsgehalt besitzt als g. Trotz der Tatsache, dass f und g verschiedene Kandidatendokumente referenzieren gehen wir davon aus, dass es unerheblich ist welche der beiden Quellen plagiiert wurde. Aus diesem Grund wird nach der Identifizierung einer dominierten Fundstelle g diese aus der Ergebnismenge entfernt und der Algorithmus mit den restlichen Kombinationen aus Fundstellen fortgesetzt.

Mithilfe der ersten drei Arbeitsschritte des *Postprocessing* werden zusammengehörige Ergebnisse vereint, solche mit zu geringem Informationsgehalt entfernt und redundante Informationen reduziert.

Filter

Dies ist der letzte Schritt des Postprocessing im Rahmen des Text Alignments. Alle Fundstellen werden nochmal angesehen und zu kurze werden herausgefiltert. Der Parameter *MinPlagLen* $\in \mathbb{N}_+$ gibt die Mindestlänge der Stellen an, die zum Schluss als verdächtige Stellen in die Ausgabemenge übernommen werden sollen.

4.6 XML-Aufbereitung der Ergebnisse

Ein Nachteil der von uns gewählten Integer-Repräsentation ist, dass die intern verwendeten Zahlen von dem eigentlichen Text nicht mehr viel beinhalten und eine direkte Zurückführung erschwert wird. Da die Ausgabe der Software nicht nur für die Software, sondern auch den Anwender lesbar sein soll, haben wir uns für ein XML-Ausgabeformat entschieden. Die Ausgabe setzt sich aus folgenden Elementen zusammen: Sämtliche Ergebnisse werden sequentiell unter dem *<results>*-Element aufgeführt. Jedes einzelne Ergebnis wird mit einem *<result>*-Element genauer beschrieben. Dieses enthält die folgenden Attribute:

- *document_id*: ist die eindeutige ID des Kandidatendokumentes, auf welches sich dieses Ergebnis bezieht.
- *candidate_filename* ist der Dateiname des Kandidatendokumentes.
- *suspicious_start_index_words* ist der Wortindex, bei dem das Plagiat beginnt.
- *suspicious_length_words* ist die Wortanzahl, aus der das Plagiat besteht.
- *suspicious_length_codepoints* ist der Beginn des Plagiats gemessen in Codepoints (Unicode).
- *suspicious_start_index_codepoints* ist die Länge des Plagiats in Codepoints (Unicode).
- *suspicious_start_index_byte* ist der Beginn des Plagiats gemessen in Bytes.
- *suspicious_length_byte* ist die Länge des Plagiats in Bytes.

Alle Attributen, welche die Stellen innerhalb der zu untersuchenden Dokumentes beschreiben, existieren analog für das Kandidatendokument (beginnend mit *candidate_*). Zur Verbesserung der Menschenlesbarkeit besitzt das *<result>*-Element zwei Subelemente *<plagiarism>* und *<original>*. Diese beinhalten im Fall von *<plagiarism>* den Text des originalen verdächtigen Dokumentes und entsprechend bei *<original>* den Text des ursprünglichen Kandidatendokumentes. Eine generierte Ausgabe könnte wie in Quelltext 4.1 aussehen. Hierbei wurden die folgenden beiden Texte überprüft:

Text 1 (Original): *Plagiarism is the wrongful appropriation and stealing and publication of another author's language, thoughts, ideas, or expressions and the representation of them as one's own original work.*

Text 2 (Plagiat): *Plagiarism is the wrongful appropriation and stealing and publication bla bla bla bla bla bla bla bla bla of another author's language, thoughts, ideas, or expressions bla bla bla bla bla bla bla bla bla and the representation of them as one's own original work.*

Die Ausgabe weist drei Funde auf, was durch die drei *<result>*-Elemente gezeigt wird. Jedes Ergebnis weist auf die selbe *document_id*, in diesem Fall 1, da nur ein Vergleichsdokument gegeben ist. Die Indizes der Plagiate innerhalb des verdächtigen Dokuments werden mittels *suspicious_start_index_words* als 0, 14 und 28 identifiziert. Analog wurden die plagiierten Stellen innerhalb des Kandidatendokuments mit den Indizes 0, 5 und 10 erkannt und durch das Attribut *candidate_start_index_words* ausgegeben. Die Plagiate werden durch den Inhalt der *<plagiarism>*-Elemente wie folgt dargestellt:

1. *Plagiarism is the wrongful appropriation and stealing and publication*
2. *author's language, thoughts, ideas, or expressions*

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <document>
3    <results>
4      <result document_id="1" suspicious_start_index_words="0"
       candidate_start_index_words="0" suspicious_length_words="5"
       candidate_length_words="5" suspicious_length_codepoints="69"
       suspicious_start_index_codepoints="0" suspicious_length_byte="69"
       suspicious_start_index_byte="0" candidate_filename="/tmp/
       candidate_document_original.txt" candidate_length_codepoints="69"
       candidate_start_index_codepoints="0" candidate_length_byte="69"
       candidate_start_index_byte="0">
5        <plagiarism>Plagiarism is the wrongful appropriation and stealing
       and publication</plagiarism>
6        <original>Plagiarism is the wrongful appropriation and stealing and
       publication</original>
7      </result>
8      <result document_id="1" suspicious_start_index_words="14"
       candidate_start_index_words="5" suspicious_length_words="5"
       candidate_length_words="5" suspicious_length_codepoints="50"
       suspicious_start_index_codepoints="117" suspicious_length_byte="50"
       suspicious_start_index_byte="117" candidate_filename="/tmp/
       candidate_document_original.txt" candidate_length_codepoints="50"
       candidate_start_index_codepoints="81" candidate_length_byte="50"
       candidate_start_index_byte="81">
9        <plagiarism>author&amp;apos;s language, thoughts, ideas, or expressions<
       /plagiarism>
10       <original>author&amp;apos;s language, thoughts, ideas, or expressions</
       original>
11     </result>
12     <result document_id="1" suspicious_start_index_words="28"
       candidate_start_index_words="10" suspicious_length_words="4"
       candidate_length_words="4" suspicious_length_codepoints="55"
       suspicious_start_index_codepoints="212" suspicious_length_byte="55"
       suspicious_start_index_byte="212" candidate_filename="/tmp/
       candidate_document_original.txt" candidate_length_codepoints="55"
       candidate_start_index_codepoints="140" candidate_length_byte="55"
       candidate_start_index_byte="140">
13       <plagiarism>representation of them as &amp;lt;one&amp;apos;s&amp;gt; own &amp;quot;
       original&amp;quot; &amp;amp;work.</plagiarism>
14       <original>representation of them as &amp;lt;one&amp;apos;s&amp;gt; own &amp;quot;
       original&amp;quot; &amp;amp;work.</original>
15     </result>
16   </results>
17 </document>
```

Quelltext 4.1: Beispiel-XML-Ausgabe bei dem Vergleich zweier sehr ähnlicher Texte.

3. *representation of them as one's own original work.*

Entsprechend werden die plagiierten Stellen durch das *<original>*-Element gekennzeichnet:

1. *Plagiarism is the wrongful appropriation and stealing and publication*
2. *author's language, thoughts, ideas, or expressions*
3. *representation of them as one's own original work.*

Wie beobachtet werden kann, hat SciencePlag die wörtliche Übernahme aus dem Kandidatendokument vollständig erkannt.

4.6.1 Erweiterung für Output Viewer

Um die XML-Ausgabe in dem im nächsten Abschnitt vorgestellten Output Viewer verwenden zu können, ist die XML-Ausgabe um einen Punkt erweitert worden. Neben den *<result>*-Tags ist ein *<meta>*-Tag der Ausgabe hinzugefügt worden. Dieses enthält ein Tag *<suspicious>* sowie eine List von *<candidate>*-Tags. Innerhalb dieser ist jeweils ein Tag *<text>* für den kompletten Text des Dokuments und *<id>*-Tags zu finden. Desweiteren jeweils Informationen zu Autor und Titel in den entsprechenden *<author>*- und *<title>*-Tags, falls diese zur Verfügung stehen.

4.6.2 HTML-Darstellung

Um die Lesbarkeit der Ergebnisse zu verbessern, haben wir das Tool *Output Viewer* entwickelt, das die Ergebnisse aus dem Kandidatendokument und verdächtigem Dokument gegenüberstellt. Ziel ist dabei eine benutzerfreundliche Darstellung zu erstellen, die es erlaubt einzelne verdächtige Stellen gegenüberzustellen und im Kontext einzusehen.

Funktionsweise

Output Viewer ist unabhängig vom SciencePlag-Programm entwickelt und kommuniziert nicht direkt mit diesem. Stattdessen wird die XML-Ausgabe zu einem verdächtigen Dokument gelesen und verarbeitet. Diese Ausgabe enthält zum einen Informationen zu den Fundstellen, bestehend jeweils aus Position und Länge, für Kandidatendokument und Quelldokument, zum anderen sind der gesamte Text und die Metadaten wie Titel, Autor der betroffenen Dokumente enthalten. Der vollständige Text wird angezeigt und mithilfe der Positionsmarkierungen mit farblichen Kennzeichnungen erweitert.
Um die Übersichtlichkeit zu gewährleisten, wird das Kandidatendokument jeweils nur mit einem Quelldokument verglichen. Der Grund dafür ist, das bei der Suche nach verdächtigen Stellen zu Überscheidungen aus unterschiedlichen Quelldokumenten kommen kann. In diesem Fall ist es optisch als auch technisch schwierig die Stelle dem entsprechenden Kandidatendokument zuzuordnen.

Technische Umsetzung

Output Viewer ist ausschließlich mit Webtechnologien entwickelt worden. Das erlaubt die Verwendung ohne vorherige Installation. Außerdem können die Ergebnisse betrachtet werden ohne SciencePlag zu installieren, da keine direkte Kommunikation stattfindet. Die Darstellung erfolgt durch Hypertext Markup Language (*HTML*) und Cascading Style Sheets (*CSS*). Als Design Framework wird dabei *angular-material* verwendet. Dieses

basiert auf dem von Google spezifierten *Material Design*.[10] Für die Implementierung der Funktionalität wird die Websprache *JavaScript* verwendet. Für die Strukturierung des Codes und die Interaktion mit dem HTML Document Object Model (*DOM*) wird *angularjs* verwendet. Um Visualisierungen zu erzeugen wird die *JavaScript* Bibliothek *chart.js* verwendet.
Eine Liste der Bibliotheken mit den entsprechenden Quellen ist im Anhang im Kapitel B zur lokalen Installation des Output Viewers zu finden. Dort ist außerdem eine Aufführung über alle Abhängigkeiten zu finden, um Output Viewer lokal zu verwenden.

[10] `https://www.google.com/design/spec/material-design/introduction.html` (Zuletzt abgerufen: 06.04.2016)

Abbildung 4.9: Ansicht von Output Viewer für eine Ausgabe der Dateien aus dem PAN-Korpus

KAPITEL 5

Benutzerschnittstelle und Verwendung der Implementierung

Wie schon in Kapitel 4 angesprochen soll SciencePlag nicht nur die automatische Entdeckung von Plagiaten ermöglichen, sondern auch das Testen und Vergleichen von Algorithmen einfach gestalten. Zu diesem Zweck besteht das Kommandozeilenprogramm SciencePlag für Linux aus mehreren Unterprogrammen oder *Subbefehlen*, die einzeln aufgerufen werden können, um die in Kapitel 4 erwähnten Komponenten möglichst auch einzeln aufzurufen und überprüfen zu können. Diese werden in den nächsten Abschnitten näher erläutert. Zuletzt wird noch erläutert, wie man mit dem Hilfsscript wikiplag.py aus der Online-Enzyklopädie Wikipedia eine Datenbasis von bereits beachtlicher Größe erzeugt.

5.1 Aufruf des Kommandozeilenprogramms

Jeder einzelne Subbefehle von SciencePlag setzt eine eigene Funktion um, wie z. B. die Erstellung einer statischen Datenbasis durch `build`. Dazu werden zu ihrer Steuerung *Programmoptionen*, kurz *Optionen*, übergeben (z. B. via `--directory PATH` das Verzeichnis PATH, in dem die Datenbasis erstellt werden soll). Die Optionen unterscheiden sich größtenteils in den verschiedenen Subbefehlen. Es gibt jedoch Gemeinsamkeiten, insbesondere in der Option `--directory`.

Eine allen Subbefehlen und SciencePlag selbst gemeinsame Option ist `--help` bzw. `-h` (siehe auch Beispiele in 5.3.1). `--help` ist ein Beispiel für eine parameterlose Option, auch *Flag* genannt. Häufiger anzutreffen sind allerdings Optionen, die auch noch zusätzliche Parameter benötigen (siehe `--directory`).

Während einige Optionen für den Aufruf von Subbefehlen notwendig sind, sind andere nur optional, da bereits Standardwerte für sie festgelegt sind. Falls Standardwerte existieren müssen, sind diese auch im Programmcode definiert, können jedoch auch aus den in Abschnitt 5.3.3 beschriebenen Konfigurationsdateien gelesen werden. Die Konfigurationsdateien ermöglichen somit neue überschreibende Standardwerte festzulegen, für die SciencePlag nicht neu kompiliert werden muss, was ein großer Vorteil beim Testen der zugrunde liegenden Algorithmen ist.

5.2 Übersicht der Subbefehle

Es gibt drei Optionen, die bei jedem Subbefehl verwendet werden können.

lang	kurz	Beschreibung
--help	-h	zeige detaillierte Informationen zum Subbefehl
--output-path	-o	Datei, in die Ausgabe des Resultats geschrieben werden soll
--config	-c	Pfad zur Konfigurationsdatei

Im Folgenden werden die einzelnen Subbefehle beschrieben. In *Erstellung und Löschung von Datenbasen* werden die Subbefehle beschrieben, die sich mit dem Aufbau und der Auflösung einer statischen Datenbasis beschäftigen. Nachdem eine Datenbasis erstellt worden ist, kann mit den Befehlen in *Plagiatssuche* nach Plagiatsstellen aus Dokumenten in der Datenbasis gesucht werden. Mit den Subbefehlen in *Informationen zu Datenbasen und Vorverarbeitung* kann man genauer analysieren, wie die Dokumente in einer Datenbasis aussehen und wie das Preprocessing auf einem Dokument arbeitet. Anschließend werden wir eine kurze Einführung in den *Interaktiven Modus* geben. Einige Beispiele zur Verwendung der Subbefehle folgen im anschließenden Abschnitt.

5.2.1 Erzeugung und Entfernung von Datenbasen

Hier sind die Befehle aufgelistet, die sich damit beschäftigen, Quelldokumente in einer statischen Datenbasis zu sammeln oder eine Datenbasis vollständig zu löschen. Die Angabe einer Datenbasis bei den Subbefehlen ermöglicht es, verschiedene Datenbasen nebeneinander zu verwenden.

`build`: Erstelle eine neue Datenbasis

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur neuen Datenbasis
--source-documents	-a	Pfade zu Quelldokumenten
oder		
--source-directories	-b	Pfade von Verzeichnissen, in denen Quelldokumente erhalten sind

Bemerkungen: Erzeugt im angegebenen Pfad eine neue Datenbasis. Dabei können einzelne Dokumente oder Pfade zu Ordnern mit Dokumenten angegeben werden, um sie als Quelldokumente zu nutzen.

`add`: Erweitere eine vorhandene Datenbasis

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur neuen Datenbasis
--source-documents	-a	Pfade zu Quelldokumenten
oder		
--source-directories	-b	Pfade von Verzeichnissen, in denen Quelldokumente erhalten sind

Bemerkungen: Funktioniert ähnlich wie `build`, verlangt jedoch eine bereits vorhandene Datenbasis. Die anderen Optionen haben die gleiche Funktion wie oben.

`delete`: Entferne eine vorhandene Datenbasis vollständig

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur zu entfernenden Datenbasis

Bemerkungen: Löst die angegebene Datenbasis auf.

5.2.2 Plagiatssuche

Zum Auffinden von Plagiaten genügt der Befehl `find`. Die anderen Subbefehlen können vor allem dafür genutzt werden, die wichtigen Teilkomponenten des Candidate Retrieval und Text Alignments auch individuell aufzurufen. Eine Besonderheit der hier genannten Subbefehle besteht darin, dass sich bei ihnen, über weitere Optionen, die intern benutzten Verfahren parametrisieren lassen. Letztere werden hier jedoch nicht aufgelistet, bei Bedarf ruft man hierfür den Subbefehl mit `--help` auf.

`find`: Finde Plagiatsstellen im verdächtigen Dokument

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur statischen Datenbasis
--suspicious-document	-s	Pfad zum verdächtigen Dokument

Bemerkungen: Findet Stellen aus dem angegebenen verdächtigen Dokument, die nach Meinung von SciencePlag aus Dokumenten in der Datenbasis plagiiert wurden. Diese Stellen werden als Gegenüberstellung von Original und Plagiat ausgegeben. Die Konfiguration der parametrisierten Verfahren kann mit besonderen Optionen angepasst werden, die sich über `--help` anzeigen lassen.

`retrieve`: Finde passende Kandidatendokumenten zum verdächtigen Dokument

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur statischen Datenbasis
--suspicious-document	-s	Pfad zum verdächtigen Dokument

Bemerkungen: Führt ausschließlich das Candidate Retrivel aus und gibt die Quelldokumente aus der Datenbasis an, die am ähnlichsten zum verdächtigen Dokument sind. Ausgabe der gefundenen Kandidatendokumente erfolgt geordnet nach dem Maß der Ähnlichkeit. Die Konfiguration der parametrisierten Verfahren kann mit besonderen Optionen angepasst werden, die sich über `--help` anzeigen lassen.

`align`: Finde Plagiatsstellen aus angegebenen Kandidatendokumenten im verdächtigen Dokument

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur statischen Datenbasis
--source-ids	-i	IDs der Quelldokumente aus der Datenbasis
--suspicious-document	-s	Pfad zum verdächtigen Dokument

Bemerkungen: Führt das Text Alignment auf dem verdächtigen Dokument und den mittels IDs spezifizierten Kandidatendokumenten aus. Die Ausgabendarstellung entspricht den Angaben in `find`. Die Konfiguration der parametrisierten Verfahren kann mit besonderen Optionen angepasst werden, die sich über `--help` anzeigen lassen.

5.2.3 Informationen zu Datenbasen und Vorverarbeitung

Die hier aufgeführten Subbefehle sind u. a. für eine Einsicht in das Preprocessing gedacht. Zusätzlich findet sich hier noch die Kurzbeschreibung zu `show`, welches die globalen Daten einer Datenbasis einschließlich der Quelldokumente anzeigt.

`preprocess`: Führe das Preprocessing auf einem Dokument durch

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur statischen Datenbasis
--source-document	-s	Pfad zum Dokument

Bemerkungen: Zeigt an, welche Informationen während des Preprocessings aus dem angegebenen Dokument gewonnen wurden. Dazu gehören der eingelesene Text im Rohformat und Ergebnisse der Normalisierung der Wörter.

`show`: Zeige eine Übersicht über die Datenbasis an

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur statischen Datenbasis

Bemerkungen: Zeigt einige Informationen zur angegeben Datenbasis an, dazu gehört die Übersicht über die einzelnen Dokumente mit der Zuordnung von ID und Titel des Quelldokuments. Die IDs werden für die Subbefehle `align` und `lookup` benötigt.

`lookup`: Zeige die in der Datenbasis gespeicherten Informationen über ein Quelldokument an

Notwendige Parameter:

lang	kurz	Beschreibung
--directory	-d	Pfad zur statischen Datenbasis
--source-id	-i	ID des Quelldokuments aus der Datenbasis

Bemerkungen: Zeigt an, welche Preprocessingdaten zu dem mit der angegebenen ID spezifizierten Quelldokument in der Datenbasis vorhanden sind. Ausgabedarstellung entspricht den Angaben in `preprocess`.

5.2.4 Interaktiver Modus

Der Subbefehl `interactive` fällt aus dem normalen Schema heraus. Er hat keine Optionen und startet nur den interaktiven Modus. Der interaktive Modus ermöglicht es, SciencePlags Subbefehle in einem persistenten SciencePlag-Prozess aufzurufen. Dazu bietet SciencePlag eine einfache Kommandozeile, in die man die Subbefehle und ihre Optionen direkt eingeben kann.

Der größte Vorteil vom interaktiven Modus besteht darin, dass man mit `load` eine bereits vorhandene Datenbasis von der Festplatte laden kann und bei den folgenden Aufrufen von `loaded SUBBEFEHL` mit dieser bereits geladenen Datenbasis arbeiten kann, was zu einer großen Beschleunigung des gesamten Vorgangs führt. Mehr Informationen über die Anwendung des interaktiven Modus findet man, wenn man in der SciencePlag-Kommandozeile den besonderen Subbefehl `--help` aufruft.

5.3 Tutorial zur Verwendung

An dieser Stelle soll der Nutzer von SciencePlag durch einige Beispiele an die Verwendung von SciencePlag herangeführt werden. Dabei werden wir weiter hinten auch die fortgeschrittenen Funktionen beschreiben, wie die Verwendung von Konfigurationsdateien und des Loggings. Für eine vollständige Dokumentation aller Möglichkeiten der Subbefehle sei auf deren `--help`-Ausgaben verwiesen. Wir gehen für diese Anleitung davon aus, dass das SciencePlag-Programm als `splag` in der Kommandozeile aufgerufen werden kann. Dies kann ohne Probleme durch einen direkten Pfad zu SciencePlag ersetzt werden.

5.3.1 Beispiele für den Arbeitsablauf

Unser Arbeitsverzeichnis für diese Anleitung soll folgendermaßen aussehen:

```
> tree
.
|-- logging.conf
|-- plag.txt
|-- tut
|   |-- forgotten.txt
|   |-- text1.txt
|   |-- text2.txt
|-- tutorial.cfg

1 directory, 6 files
```

Unser Ziel ist es zu überprüfen, die Datei PLAG.TXT Plagiate enthält.

Erstellung einer Datenbasis

Um mit SciencePlag zu überprüfen, ob das verdächtige Dokument PLAG.TXT Plagiate enthält, muss zunächst eine statische Datenbasis erzeugt werden. SciencePlag findet nur Plagiate von Dokumenten, die in einer Datenbasis vorhanden sind, um möglichst viel von Vorverarbeitungsschritten zu profitieren.

Um eine Datenbasis zu erstellen, wird der Subbefehl `build` genutzt. `build` erwartet beim Aufruf ein Verzeichnis, in dem die Datenbasis erstellt werden soll (--directory) und zusätzlich ein oder mehrere Quelldokumente (`--source-documents`) und/oder Verzeichnisse von Quelldokumenten (`--source-directories`).

```
> splag build

build: build a new database with specified source documents
Cmdline options:
  -h [ --help ]                    show detailed information about
                                   possible parameters
  -c [ --config ] arg              path 'arg' to configuration file

  -o [ --output-path ] arg         if given, 'arg' is a path to a file
                                   where output should be written into
Required options:
```

```
  -d [ --directory ] arg           directory 'dir' of new database
  -a [ --source-documents ] arg    paths 'arg' to source documents
  -b [ --source-directories ] arg  add all files of directory 'arg'
                                   (non-recursive) to database
```

Die ersten drei Optionen sind in allen Subbefehlen außer `interactive` verfügbar. Mehr zur Option `--config` findet man in Abschnitt 5.3.3, `--output-path` ist insbesondere für die forgeschrittenen Funktionen sinnvoll. Die Option `--help` wurde schon in der Einführung erwähnt und gibt detailliertere Informationen über den entsprechenden Subbefehl aus. Im Fall von `build` sieht die Ausgabe so aus:

```
> splag build --help

build: build a new database with specified source documents
splag build -- Allowed options:

Cmdline options:
  -h [ --help ]                   show detailed information about
                                  possible parameters
  -c [ --config ] arg             path 'arg' to configuration file

Required options:
  -d [ --directory ] arg              directory 'dir' of new database
  -a [ --source-documents ] arg       paths 'arg' to source documents
  -b [ --source-directories ] arg     add all files of directory 'arg'
                                      (non-recursive) to database
Parameter options:
  --pre.language arg              language for preprocessing:
                                  'en'(default) or 'de'

Logging options:
  --log.config arg                path 'arg' to configuration file
                                  for logging
```

Abschitt 5.3.5 wird sich genauer mit dem Logging beschäftigen.

Da wir im Folgenden mit deutschen Texten arbeiten wollen, müssen wir bei der Erstellung der Datenbasis mit `--pre.language de` explizit angeben, dass diese auf die deutsche Sprache angepasst werden soll:

```
> splag build --pre.language de --directory ./tut_db
    --source-documents "tut/text1.txt" "tut/text2.txt"

build: build a new database with specified source documents
database directory: ./tut_db
single source documents specified: tut/text1.txt; tut/text2.txt;
directories with source documents:
all source documents: tut/text1.txt; tut/text2.txt;
preprocessing language: de
preprocessor module: cpp
```

```
path ./tut_db does not exist
Initialize new database...
name:        tut_db
location:        .
sqlite database successfully created and opened
DocumentBase: DocumentBase successfully created at ./tut_db
1: Adding "tut/text1.txt"...
2: Adding "tut/text2.txt"...
Adding new source documents to database...
Adding 2 elements to database.
Adding next 2 elements.
Finished adding elements to DB.
2
Saving wordmapping...
Finishing...
```

Im diesem Einführungsbeispiel haben wir nachträglich ein weiteres Dokument FORGOTTEN.TXT gefunden, welches wir noch in unsere Datenbasis einfügen wollen. Diese Aufgabe können wir auf zwei Arten lösen. Die erste Option ist der komplette Neuaufbau der Datenbasis mit `build`, nachdem die alte Version mit `delete` zunächst gelöscht worden ist.

```
> splag delete -d "./tut_db"
[...]
> splag build -d --pre.language de "./tut_db" -a "tut/text1.txt"
  "tut/text2.txt" "tut/forgotten.txt"
[...]
```

An dieser Stelle haben wir auf die Kurzversionen der Optionen zurückgegriffen. Alternativ können hier wir auch `--source-directories tut` anstelle von `-a [...]` verwenden. So wird das gesamte Verzeichnis TUT in die Datenbasis aufgenommen.

Einfacher geht das hier allerdings mit dem Subbefehl `add`, das in ähnlicher Weise wie `build` aufgerufen wird und zu einer existierenden Datenbasis die angegebenen Dokumente hinzufügt.

```
> splag add --directory "./tut_db" --source-documents
    "tut/forgotten.txt"
[...]
```

`add` erwartet in `--directory` – wie alle anderen Subbefehle ausgenommen von `build` – eine bereits vorhandene Datenbasis, mit der gearbeitet wird.

Weiterhin kann mit `show` angezeigt werden, welche Quelldokumente sich in einer Datenbasis befinden:

```
> splag show --directory "./tut_db"

show: show source documents in database
database directory: ./tut_db
```

```
Load existing database...
name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Print summary...
path to database: ./tut_db
language: de
number of documents: 3

id        author        title
0        UNKNOWN        text2
1        UNKNOWN        text1
2        UNKNOWN        forgotten
Finishing...
```

Von besonderem Interesse sind die IDs der einzelnen Dokumente. Diese IDs werden in den Befehlen `lookup` und `find` benötigt, doch dazu später mehr.

Damit wäre die Einleitung zur Erstellung, Erweiterung, Einsicht und Löschung von Datenbasen beendet. Mit unserer angelegten Datenbasis können wir nun im nächsten Abschnitt nachprüfen, ob PLAG.TXT plagiierte Stellen enthält.

Testen auf Plagiate

Mit unserer bereits mit passenden Quelldokumenten gefüllten Datenbasis können wir nun auch unser verdächtiges Dokument mit `find` darauf prüfen, ob es Plagiate aus diesen Quelldokumenten enthält.

```
> splag find --directory "./tut_db" --suspicious-document "plag.txt"

find: find plagiarized passages from documents in database in
    suspicious document
database directory: ./tut_db
suspicious document: plag.txt
cr.filter = similarity
cr.number = 1
cr.similarity = 0.3
ta.min-lcp = 3
ta.delta-susp = 3
ta.delta-cand = 3
ta.common-min = 10
ta.plag-min-len = 5

Load existing database...
name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Preprocess suspicious document...
```

```
Entering Candidate Retrieval...
Entering Text Alignment...
Print plagiarized passages...
<?xml version="1.0" encoding="utf-8"?>
<document>
  <results>
    <result document_id="0" [...]>
      <plagiarism>letzte Satz ist ganz besonders einfach zu
plagiieren, da besondere Worte wie &quot;Desoxyribonukleinsäure&quot;
und &quot;Graphtheorie&quot; vorkommen.
</plagiarism>
      <original>letzte Satz ist ganz besonders einfach zu
plagiieren, da besondere Worte wie &quot;Desoxyribonukleinsäure&quot;
und &quot;Graphtheorie&quot; vorkommen.
</original>
    </result>
  </results>
  <meta>
    [...]
  </meta>
</document>
Finishing...
```

Die Ausgabe ist ein XML-Dokument, welches das von SciencePlag entdeckte Plagiat dem Originaltext gegenüberstellt und einige Zusatzinformationen enthält, die für Vorführzwecke hier zunächst entfernt wurden. Um diese Stellen in ihrem Kontext auch wieder zu finden, sind auch die Positionen enthalten, an denen sie in den Texten gefunden werden können. Alternativ kann auch GUIPlag eine angenehmere Darstellung der Funde erzeugen, dazu mehr in Abschitt 4.6.2.

In obiger Ausgabe befinden sich außerdem Informationen über die Werte der Algorithmus-internen Parameter, die bei Belieben auch noch angepasst werden können. Über die `--help`-Funktionalität von `find` lassen sich deren Kurzbeschreibungen ausgeben:

```
> splag find --help

[...]
Parameter options:
  --cr.filter arg              selection criterion: 'constant' number
                               of most similar documents or all
                               documents with minimal 'similarity'
  --cr.number arg              number of candidate documents (if
                               filter == 'constant')
  --cr.similarity arg          minimal cosine-similarity of suspicious
                               and candidate documents (if filter ==
                               'similarity')
  --ta.min-lcp arg             minimal length of longest common prefix
                               for suffix array search
  --ta.delta-susp arg          maximal distance for merging adjacent
                               passages in the suspicious document
```

```
  --ta.delta-cand arg          maximal distance for merging adjacent
                               passages in a candidate document
  --ta.common-min arg          minimal number of occurences to be
                               considered as a standard phrase
  --ta.plag-min-len arg        minimal length of a result after
                               postprocessing
[...]
```

Die Parameter sind hier den Modulen des Candidate Retrievals (*cr*) und des Text Alignments (*ta*) zugeteilt. Da hieraus nicht direkt ersichtlich ist, was genau im Hintergrund passiert, wird dem Leser dazu geraten, sich bei Interesse die Beschreibungen der einzelnen Komponenten in Kapitel 4 genauer anzusehen.

Mit folgender Parametrisierung kann erzwungen werden, dass das Candidate Retrieval nur ein verdächtiges Dokument an das Text Alignment zur Überprüfung übergibt, es werden also nur Plagiatsstellen aus der wahrscheinlichsten Quelle angegeben:

```
> splag find --directory "tutorial" --suspicious-document "plag.txt"
  --cr.k 1
[...]
```

Wie in Kapitel 6 noch gezeigt wird, kann die Parametrisierung starken Einfluss auf die angegebenen vermeintlichen Plagiate enthalten.

An dieser Stelle haben wir die wichtigsten Funktion von SciencePlag erläutert. Der Leser kann jetzt eine Datenbasis erzeugen und Plagiate entdecken. Die nächsten Abschnitte behandeln fortgeschrittene Themen wie das Überspringen von Teilschritten bei der Plagiatsfindung, Konfigurationsdateien und Loggingeinstellungen. Sie sind vor allem nützlich bei Verbesserungsversuchen von SciencePlag. Davor werden wir aber noch eine kurze Einführung in Output Viewer (siehe Abschnitt 4.6.2) geben, der es ermöglicht, von SciencePlag vermutetet Plagiate mitsamt ihrem Kontext graphisch angemessen gegenüberzustellen.

5.3.2 Graphische Repräsentation der Ergebnisse

Nach dem Start von Output Viewer gemäß Abschnitt B im Anhang kann er in einem beliebigen Browser unter der URL `http://localhost:9000/#/compare/splag` aufgerufen werden. Nach dem Laden der Seite wird der Benutzer aufgefordert die XML-Ausgabe von SciencePlag auf eine von zwei Arten einzugeben. Die erste Möglichkeit besteht darin, die Ausgabe von `find` oder `align` über eine abgespeicherte Datei an Output Viewer weiterzugeben. Zu diesem Zweck kann die Option `--output-path` in SciencePlag benutzt werden, mit der angegeben wird, dass der XML-Output in dem angegeben Pfad gespeichert werden soll. Alternativ kann die XML-Ausgabe von `find`/`align` auch direkt aus der Kommandozeile kopiert und in den Dialog hineinkopiert werden.

Mit der Bestätigung wird die Ansichtsseite geöffnet. Diese ist horizontal in zwei Bereiche aufgeteilt. Auf der linken Seite befindet sich das verdächtige Dokument und auf der rechten das aktuell ausgewählte Quelldokument. Jeweils über dem Text sind Autor und Titel zu finden, sofern diese Informationen in SciencePlag vorhanden waren. Neben der textuellen Auswertung sind unter dem Punkt *Statistik* noch zwei Grafiken zu finden. Die erste stellt die Verteilung der Fundstellen auf die gefundenen Quelldokumente als Kreisdiagramm dar. Die zweite zeigt die Anzahl der Fundstellen pro Seite als Säulendiagramm. Da die Länge einer Seite nicht einheitlich ist oder gelegentlich eine Einteilung auf Seiten

nicht möglich ist, z. B. bei Wikipedia, wird jeweils von der Normseite mit 1500 Anschlägen ausgegangen.

Testen einzelner Komponenten von SciencePlag

Im Folgenden wird erläutert was wir unternehmen können, wenn ein unbefriedigendes Ergebnis durch die falsche Auswahl der Kandidatendokumente entsteht. Ist es möglich zu überprüfen, ob die richtigen Kandidatendokumente gefunden wurden? Kann man das Candidate Retrieval vielleicht einfach überspringen, indem die Kandidatendokumente direkt angegeben werden und nur noch das Text Alignment durchgeführt wird? Findet sich der Fehler doch noch an einer früheren Stelle, wie z.B. dem Preprocessing?

Bei der Beantwortung dieser Fragen zeigt sich das modulare Design von SciencePlag sehr nützlich. Durch entsprechende Subbefehle können spezifische Komponenten der Software getestet werden, sodass sich die Fragen im letzten Absatz einfach beantworten lassen.

Zunächst können wir uns ansehen, wie die Dokumente intern repräsentiert werden, indem wir uns Informationen des Preprocessings ausgeben lassen. Eine solche Funktion führen zwei Subbefehle aus: `preprocess` für Dokumente, die uns als Datei gegeben sind, und `lookup` für die Quelldokumente in einer erstellten Datenbasis.

Wir beginnen mit `preprocess`. Hier muss der Benutzer die Datei direkt angeben, von der er die Ergebnisse der Vorverarbeitung sehen wollen. Da die Ergebnisse des Preprocessings direkt von einigen Daten einer Datenbasis abhängen, insbesondere der Sprache, muss auch diese explizit angegeben werden:

```
> splag preprocess --directory "./tut_db" --source-document "plag.txt"

preprocess: preprocess input document and only print output
database directory: ./tut_db
source document: plag.txt

Load existing database...
name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Preprocess document...Print preprocessed document...

[title]_____________
SUSP_TITLE
[author]____________
SUSP_AUTHOR
[original_text]______
Hier {gibt} es ein {wundervolles} {Plagiat} zu {bestaunen}, das aus
{mehreren} {Texten} {entnommen} {wurde}, ohne {korrekt} zu {zitieren}.
{Hoffentlich} {fällt} "{Splag}" nicht auf, {dass} hier {eigentlich} nur
abgeschrieben} {wurde}.
Man kann nur {beten} und {hoffen}, {dass} es diesem {brillianten}
{Programm} nicht {auffällt}.
Dieser {letzte} {Satz} ist {ganz} {besonders} {einfach} zu
{plagiieren}, da {besondere} {Worte} wie "{Desoxyribonukleinsäure}"
```

```
und "{Graphtheorie}" {vorkommen}.
[preprocessed text]__
(14)(15)(44)(17)(45)(46)(47)(48)(19)(49)(5)(6)(50)(8)(9)(10)(48)(21)
(23) (8)(51)(52)(38)(24)(25)(26)(27)(28)(29)(27)(30)(31)(32)(33)
[start positions and lengths]__
(5,4)(17,12)(30,7)(41,9)(60,8)(69,6)(76,9)(86,5)(98,7)(109,8)(119,11)
(131,6)(139,5)(157,4)(167,10)(182,13)(196,5)(217,5)(227,6)(235,4)
(250,11)(262,8)(277,9)(295,6)(302,4)(311,4)(316,9)(326,7)(337,10)
(352,9)(362,5)(373,23)(403,12)(417,9)
Finishing...
```

Auf ähnliche Weise können wir `lookup` verwenden, um ein bereits vorverarbeitetes Dokument in einer Datenbasis genauer anzuschauen. Dafür benötigen wir die ID dieses Quelldokumentes, z. B. ID 0 von TEXT2.TXT, die wir weiter oben durch den Aufruf von `show` gewonnen haben:

```
> splag lookup --directory "./tut_db" --source-id 0

lookup: print preprocessing information about source document in
    database
database directory: ./tut_db
source document's ID: 0

Load existing database...
[...]
Load preprocessed document...
Print preprocessed document...

[title]_______________
text2
[author]______________
UNKNOWN
[original_text]______
Hier {gibt} es eine {wundervolle} {Abgabe} zu {bestaunen}, aber
{leider} ohne {korrekte} {Zitate}. Man kann nur {beten}, und
{vielleicht} noch {hoffen}, {dass} sie nicht {abgeschrieben} ist!
Dieser {letzte} {Satz} ist {ganz} {besonders} {einfach} zu
{plagiieren}, da {besondere} {Worte} wie "{Desoxyribonukleinsäure}"
und "{Graphtheorie}" {vorkommen}.
[preprocessed text]__
[...]
[start positions and lengths]__
[...]
Finishing...
```

Weitere hilfreiche Testbefehle sind `retrieve` und `align`, die es uns ermöglichen, nur das Candidate Retrieval oder das Text Alignment aufzurufen und die andere Komponente

zu umgehen Dabei können ähnlich wie in `find` die entsprechenden internen Parameter durch Optionen modifiziert werden.

Mit `retrieve` finden wir heraus, welche Quelldokumente für SciencePlag als Kandidatendokumente in Frage kommen. Damit kann ein Tester herausfinden, ob mit den angegebenen Parametern die richtigen Dokumente tatsächlich gefunden werden oder ob der Algorithmus oder die Parameterwerte angepasst werden müssen.

```
> splag retrieve --directory "./tut_db" --cr.filter constant
  --cr.number 2 --suspicious-document "plag.txt"

retrieve: retrieve candidate documents from database for suspicious
    document
database directory: ./tut_db
suspicious document: plag.txt
cr.filter = constant
cr.number = 2
cr.similarity = 0.3

Load existing database...
name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Preprocess suspicious document...
Entering Candidate Retrieval...
Print candidate documents...
0: UNKNOWN -- text2
1: UNKNOWN -- text1
Finishing...
```

Zusätzlich wurde hier die Anzahl der Kandidatendokumente auf 2 beschränkt.

Es ist auch möglich, das Candidate Retrieval zu überspringen und die Kandidatendokumente direkt anzugeben. So kann dann sichergestellt werden, dass man den zugrunde liegenden Algorithmus für das Text Alignment ohne den potentiellen Störfaktor von falscher Dokumentenwahl durch das Candidate Retrieval testen kann. Um Kandidatendokumente zu spezifizieren, werden auch hier wieder die IDs aus der Datenbasis benötigt (`show`).

```
> splag align --directory "./tut.db" --suspicious-document "plag.txt"
  --source-ids 0 1

align: find plagiarized passages from candidate documents in suspicious
    document
database directory: ./tut_db
source document IDs: 0 1
suspicious document: plag.txt
ta.min-lcp = 3
ta.delta-susp = 3
ta.delta-cand = 3
ta.common-min = 10
```

```
ta.plag-min-len = 5

Load existing database...
name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Preprocess suspicious document...
Loading candidate documents...
Entering Text Alignment...
Print plagiarized passages...
<?xml version="1.0" encoding="utf-8"?>
<document>
  <results>
    [...]
  </results>
  <meta>
    [...]
  </meta>
</document>
Finishing...
```

Hier haben wir uns für einen Vergleich mit den Quelldokumenten entschieden, die wir als TUT/TEXT1.TXT und TUT/FORGOTTEN.TXT in die Datenbase hinzugefügt haben.

Damit wissen wir jetzt, wie alle Subbefehle funktionieren, sowohl diejenigen, welche einfach pragmatisch unser Plagiatsergebnis ausgeben, als auch diejenigen, welche wir zum Testen der einzelnen Dokumente verwenden können. Möglicherweise hat sich der Leser auch gefragt, ob wir uns nicht weiter oben zu viele redundante Arbeit gemacht haben, indem wir immer wieder die selben Optionen angegeben haben. Die nächsten beiden Abschnitte befassen sich daher mit Konfigurationsdateien und dem interaktiven Modus von SCIENCEPLAG, der das erneute Laden ein- und derselben Datenbasis umgeht.

5.3.3 Konfigurationsdateien

In den meisten Fällen sollte die History-Funktion der verwendeten Shell zum Anpassen der gewählten Optionen genügen. Was ist aber, wenn wir sehr lange Pfade übergeben müssen? Oder wenn wir sehr viele Parameteroptionen angeben wollen? Kann man es vermeiden `--directory` immer wieder explizit anzugeben, obwohl die Option für jeden Subbefehl benötigt wird? Eine unübersichtlicher SCIENCEPLAG-Aufruf ist z. B. hier angegeben:

```
> splag find --d "pfad/zu/example" --s "dateien/DoktorarbeitGossen.txt"
    "ueberall/wie_man_einen_zwischenbericht_schreibt.txt"
    "sehr/tiefe/verschachtelung/doc12354215.txt" "verstreut/file1.txt"
    --cr.number 25 --ta.min-lcp  --ta.delta-susp 4 --log.stdout
[...]
```

Um häufig verwendete Optionen nicht immer angeben zu müssen, können daher Konfigurationsdateien geschrieben und mehrfach wiederverwendet werden können. Wie diese aufgebaut sind, kann in der Beispielkonfiguration in Abbildung 5.1 betrachtet werden. Zei-

```
# Beispielkonfigurationsdatei
directory= ./tut_db
source-documents= tut/text1.txt
source-documents= tut/text2.txt
source-documents= tut/forgotten.txt
#source-directories= tut
suspicious-document= plag.txt
[pre]
language= de
[cr]
number= 2
# alternativ auch "number.k = 2"
[ta]
#delta-cand= 5
#common-min= 10
[log]
config= logging.conf
```

Quelltext 5.1: Konfigurationsdatei TUTORIAL.CFG

lenkommentare werden mit dem Rautezeichen # eingeleitet. Die Optionen werden über Zuweisungen (=) angegeben. Falls bei einer Option mehrere Parameter erlaubt sind, muss jede einzelne Eingabe in eine neue Zeile geschrieben werden (wie bei `--source-documents` im Beispiel). Da einige Optionen sogar hierarchisch aufgebaut sind — wie man an Punkten wie z. B. an *cr.k* und *ta.delta* erkennt — ist es auch erlaubt, diese entsprechend auch auf folgende Weise zu notieren: Das Modul in eckigen Klammern, die entsprechenden Optionen diese Modules dann darunter (siehe die Zeilen unter `[cr]` und `[ta]`).

Die Verwendung einer geschriebenen Konfigurationsdatei ist einfach. Dazu wird einfach die gewünschte Konfigurationsdatei mit der Option `--config` übergeben, und SciencePlag übernimmt dann einfach die dort angegebenen Parameter:

```
> splag build -c "tutorial.cfg"
[...]
```

Hier haben wird die vorige Datenbasis gelöscht und direkt wieder mit dem selben Namen und allen uns interessierenden Quelldokumenten erstellt. Für SciencePlag macht es dabei keinen Unterschied, ob wir mit die Optionen manuell oder per Konfigurationsdatei angegeben haben. Es soll auch bemerkt werden, dass die verwendeten Pfade in Konfigurationsdateien relativ zum aktuellen Arbeitsverzeichnis liegen.

Entsprechend einfach ist es auch, die selbe Konfigurationsdatei zu nutzen, um die Beispiele von oben zu reproduzieren, indem wir noch die fehlenden Optionen ergänzen:

```
> splag find -c "tutorial.cfg" -cr.number 3
[...]
> splag align -c "tutorial.cfg" --source-ids 1 3
[...]
```

Gelegentlich kann es aber auch notwendig sein, dass man den Optionsparameter vom in der Konfigurationsdatei angegeben Wert temporär umändern möchte. Dafür muss man nicht sofort die Konfigurationsdatei wieder umschreiben, es genügt auch, durch Verwendung der Option in der Kommandozeile den Wert aus der Konfigurationsdatei für den entsprechenden Aufruf zu überschreiben, wie z. B. in `find` geschehen. Insgesamt gilt die

folgende Priorisierung für die Angabe von Optionen (eine kleinere Zahl bedeutet eine höhere Priorität):

1. direkt über Kommandozeile angegebener Wert
2. in Konfigurationsdatei angegebener Wert
3. bei optionalen Optionen: Standardwert von SciencePlag

Damit ist die Beschreibung der Verwendung von Konfigurationsdateien abgeschlossen. Wenn aber inbesondere die verwendete Datenbasis immer die selbe ist, sollte auch zusätzlich der interaktive Modus von SciencePlag genutzt werden, dessen Verwendung die Plagiatssuche auch noch spürbar schneller macht.

5.3.4 Interaktiver Modus

SciencePlag ist so konzipiert, dass die Subbefehle voneinander unabhängig in der Kommandozeile aufgerufen werden können. Das heißt außerdem, dass bei jedem Aufruf von SciencePlag ggf. die gleiche Datenbasis wiederholt geladen wird. Wenn man also zehn verschiedene Dokumente prüfen oder zehn verschiedene Parametrisierungen testen will, wird die Datenbasis auch zehn mal geladen. Da Zugriffe auf Festplatten so langsam sind, stellt das einen signifikanten Flaschenhals für die Performanz dar, der bei schon wenigen tausend Elementen deutlich auffällt.

Der interaktive Modus wird dabei helfen, das Problem in den Griff zu bekommen. Dazu startet wir ihn zunächst mit dem Subbefehl `interactive`. Er bietet dazu eine eigene Kommandozeile an, über die spezielle Befehle eingegeben werden können, die mit `help` einsehbar sind:

```
> splag interactive
interactive: start an interactive session; possibility to keep database
    loaded for operations
Starting Splag Interactive Mode...
splag > help
Special commands in Interactive Mode:
  help          show this information text
  quit, exit    quit Splag Interactive Mode
  load          load a database for further use
  build-load    build a database and keep it loaded
  unload        forget the loaded database
  loaded COMMAND    COMMAND (see following list) will be called with
                    loaded database
Other commands (as in Non-Interactive Mode):
  build, add, delete, preprocess, show, lookup, retrieve, align, find
  ('build' and 'delete' do not work in 'loaded' mode)
For further information about COMMAND, enter 'COMMAND --help'.

splag > _
```

Subbefehle wie `show` und `find` kennen wir schon und werden hier auch nicht anders aufgerufen:

```
splag > show -d ./tut_db
show: show source documents in database
```

```
database directory: ./tut_db

Load existing database...
name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Print summary...
[...]
Finishing...

splag > find -d ./tut_db -s plag.txt
find: find plagiarized passages from documents in database in
    suspicious document
database directory: ./tut_db
suspicious document: plag.txt
cr.filter = similarity
cr.number = 1
cr.similarity = 0.3
ta.min-lcp = 3
ta.delta-susp = 3
ta.delta-cand = 3
ta.common-min = 10
ta.plag-min-len = 5

Load existing database...
name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Preprocess suspicious document...
Entering Candidate Retrieval...
Entering Text Alignment...
Print plagiarized passages...
[...]
Finishing...

splag > _
```

Man beachte, dass für die Ausführung von `find` die selbe Datenbasis neu geladen wird, die wir davor schon für `show` verwendet haben. An dieser Stelle kommt die besondere Funktionalität des interaktiven Modus zu tragen. Mit dem besonderen Subbefehel `load` kann eine Datenbasis persistent in den Arbeitsspeicher geladen werden, um sie in den darauf folgenden Subbefehlen direkt anzusprechen. Dafür wird an die bekannten Subbefehle nur ein `loaded` vorgestellt:

```
splag > load -d ./tut_db
load: load an existing database and use it for following 'load'
    subcommands (interactive mode only)
```

```
database directory: ./tut_db

name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Database will be kept loaded!

splag[loaded] > loaded show
loaded: show source documents in database
[...]
Use loaded database...
[...]
Finishing...

splag[loaded] > loaded find -s plag.txt
loaded: find plagiarized passages from documents in database in
    suspicious document
[...]
Use loaded database...
[...]
Finishing...

splag > _
```

Wir interessieren uns hier insbesondere für die Ausgabe, dass eine bereits geladene Datenbasis genutzt wird.

Um eine andere Datenbasis persistent zu verwenden, muss die momentan geladene zuerst mit `unload` ordnungsgemäß aus dem Hauptspeicher verdrängt werden. Um den interaktiven Modus schließlich zu verlassen, genügt ein `exit` oder `quit`. Dabei wird eine evtl. noch geladene Datenbasis implizit mit `unload` vergessen:

```
splag[loaded] > quit
Forget loaded database...
Forget what?
Splag Interactive Mode exits...
```

Damit ist die Einführung in die anwendungsspezifischen Bestandteile von ScicencePlag abgeschlossen. Zuletzt gehen wir jedoch noch auf das Logging ein.

5.3.5 Logging in SciencePlag

Logging ist dann nützlich, wenn wir uns nicht nur für die Ausgabe eines Subbefehls interessieren, sondern auch für die Zwischenergebnisse oder -zustände unserer Komponenten. Bei jedem Aufruf von SciencePlag wird automatisch geloggt und die Ausgabe in eine bestimmte Datei beschrieben, dessen Ort und Name wie auch andere Logging-Einstellungen mit einer gesonderten Konfigurationsdatei zu setzen sind. Die zu verwendete Konfigurationsdatei kann für die Subbefehle über die Option `--log.config` angegeben werden. Wenn sie nicht explizit angegeben wird, wird die Konfigurationsdatei aus *$HOME/.splag* genutzt

```
1  -- STANDARD
2  * GLOBAL:
3    ENABLED = true
4    TO_STANDARD_OUTPUT = false
5    TO_FILE = true
6    FILENAME = "./logs/splag.log"
7    FORMAT =
8          "%datetime{%h:%m:%s:%g} [%logger] %level [%fbase %line] %msg"
9  -- PRE
10 * GLOBAL:
11   ENABLED = true
12   TO_STANDARD_OUTPUT = false
13   TO_FILE = true
14   FILENAME = "./logs/pre.log"
15   FORMAT =
16         "%datetime{%h:%m:%s:%g} [%logger] %level [%fbase %line] %msg"
17 -- CR
18 * GLOBAL:
19      [...]
20 -- ALIGN
21 * GLOBAL:
22      [...]
23 -- SPLAG
24 * GLOBAL:
25      [...]
26 [...]
```

Quelltext 5.2: Konfigurationsdatei TUTORIAL.CFG

In der Konfigurationsdatei können für jedes Logging-Modul mehrere Einstellungen vorgenommen werden (siehe auch Abb. 5.2). Wenn man sich nicht gerade auf Fehlersuche befindet, ist es am sinnvollsten, die `ENABLES`-Option aller Module auf `false` zu setzen, und das Logging ganz abzuschalten. Durch unnötiges Logging kann es nämlich sonst schnell passieren, dass sich mehrere Gigabyte Logging-Ausgaben auf der Festplatte ansammeln. Ähnlich selbsterklärend sind auch die anderen Optionen gestrickt: `TO_STANDARD_OUTPUT` gibt an, ob die Logging-Ausgaben unter die normalen SciencePlag-Ausgaben gemischt werden sollen, `FILE_NAME` bestimmt den Speicherort der Logging-Ausgaben, falls mit `TO_FILE` die Ausgabe in Dateien aktiviert ist. `FORMAT` erwartet eine Beschreibung des Loggingformats, das der Logger verwenden soll.

Mit eingeschalteter Kommandozeilenausgabe könnte die Ausgabe des Text Alignment-Loggings z. B. so aussehen:

```
splag align -c tutorial.cfg --log.config logging.conf --source-ids 0

align: find plagiarized passages from candidate documents in suspicious
    document
configuration file: tutorial.cfg
database directory: ./tut_db
source document IDs: 0
suspicious document: plag.txt
ta.min-lcp = 3
ta.delta-susp = 3
ta.delta-cand = 3
```

```
ta.common-min = 10
ta.plag-min-len = 5

Load existing database...
name:        tut_db
location:        .
sqlite database opened
DocumentBase: DocumentBase at ./tut_db successfully opened
Preprocess suspicious document...
Loading candidate documents...
Entering Text Alignment...
10:04:07:365 [ALIGN] INFO  [TextAligner.cc 11] AlignText called with 1
    candidate documents.
10:04:07:365 [ALIGN] INFO  [SuffixArrayManager.cc 24] Construct suffix
    array
10:04:07:365 [ALIGN] INFO  [SuffixArrayManager.cc 31] Expand text to
    match dc3 algorithm requirements
10:04:07:365 [ALIGN] INFO  [SuffixArrayManager.cc 44] Call dc3
    algorithm
10:04:07:365 [ALIGN] INFO  [SuffixArrayManager.cc 53] Compute lcp array
10:04:07:365 [ALIGN] INFO  [NativeAligner.cc 38] ComputeResults(..)
    called: Length of suffix array is 102
[...]
10:04:07:365 [ALIGN] INFO  [OutputGenerator.cc 34] Creating result
    element 3
<?xml version="1.0" encoding="utf-8"?>
<document>
    [...]
</document>
Finishing...
```

Nun haben wir mit der Besprechung der Loggingeinstellungen unsere Einführung zu SciencePlag beendet. Es folgt abschließend eine kurze Einführung in das Script wikiplag.py, das die reibungslose Erstellung einer Datenbasis auf Grundlage von Wikipedia ermöglicht.

5.4 Script zur Vorbereitung von Wikipedia als Datenbasis

Wikipedia[1] ist eine etablierte Online-Enzyklopädie, die besonders viel Wert darauf legt, für jeden frei zugänglich zu sein. Wikipedias Artikel sind daher unter der *Creative Commons Attribution-ShareAlike 3.0 Unported*[2], kurz *CC-by-sa-3.0*, lizensiert, was u. a. bedeutet, dass jeder Kopien der Artikel unverändert oder angepasst kopieren und weitergeben kann, solange auf die Quelle hingewiesen wird und man die Kopien auch wieder unter CC-by-sa-3.0 lizensieren lässt. Ebenfalls besteht Wikipedia vielmehr aus verschiedenen mehr oder weniger unabhängigen Unterenzyklopädien in verschiedenen Sprachen. Daher bietet sich Wikipedia als eine einfache, aber umfangreiche Sammlung von Quelldokumenten an, auf

[1] `https://www.wikipedia.org/` (Zuletzt abgerufen: 06.04.2016)
[2] `http://creativecommons.org/licenses/by-sa/3.0/` (Zuletzt abgerufen: 06.04.2016)

deren Grundlage sich eine große Datenbasis erstellen lässt. Zum einen lässt sich damit das ein oder andere Schülerplagiat abfangen, zum anderen sind allerdings auch schon bei renommierten Verlagen durchaus einige Werke veröffentlicht worden, bei denen ein großer Anteil einfach aus Wikipedia kopiert wurde[3].

Wikipedia bietet dafür selbst archivierte Versionen ihrer Artikel[4] an, im Folgenden auch *Dumps* genannt. Diese Dumps werden regelmäßig basierend auf den aktuellen Wikipedia-Artikeln für jede Sprache, zu dem eine Wikipediaversion existiert, erstellt und online zur Verfügung gestellt.

Wir interessieren uns hier besonders für die Dumps der deutschen und der englischen Wikipedia-Artikel. Unser Script WIKIPLAG.PY, für dessen Verwendung eine Installation von Python 2.7 vorhanden sein muss, ist in der Lage, den Dump in der gewünschten Sprache herunterzuladen und in eine Form zu überführen, die in eine SCIENCEPLAG-Datenbasis hinzugefügt werden kann. Eine so erzeugte Datenbasis kann dann unter der Lizenz CC-by-sa-3.0 mit einem Verweis auf `http://creativecommons.org/licenses/by-sa/3.0/` weitergegeben werden.

Verwendung des Scripts

WIKIPLAG.PY ermöglicht es, zwei verschiede Aufgaben durchzuführen. Zunächst kann mit dem Subbefehl `download` unter Angabe der Sprache mit `--language de` oder `--language en` die archivierten Dumps der deutschen bzw. englischen Wikipedia-Artikel heruntergeladen werden. Mit `--help` erfährt man außerdem analog zu SCIENCEPLAG mehr über die Anwendung des Scripts. Mit dem zusätzlichen Flag `--preprocess` kann auch sofort im Anschluss die Aufbereitung der Archive für die Verwendung mit SCIENCEPLAG durchgeführt werden, sodass man die manuelle Verwendung des anderen Subbefehl `preprocess` auslassen kann.

Der zweite Subbefehl `preprocess` hat die Funktion, bereits heruntergeladene Dumps für die weitere Verwendung in SCIENCEPLAG vorzubereiten und bietet einige hilfreiche Zusatzeinstellungen. Zur genauen Anwendung sei auf die Ausgabe von `--help` verwiesen, da in den meisten Fällen der Aufruf von `download --preprocess` ausreicht.

Mit den entpackten Dumps kann wie schon in Kapitel 5.3 erläutert mit `build` eine SCIENCEPLAG-Datenbasis erstellt werden.

[3] `http://www.spiegel.de/kultur/literatur/plagiatsvorwurf-per-facebook-c-h-beck-prueft-buch-von-bader-karsten-a-965808.html`

[4] `https://dumps.wikimedia.org/` (Zuletzt abgerufen: 06.04.2016)

KAPITEL 6

Evaluation

In diesem Kapitel widmen wir uns der Qualitätsanalyse unserer Plagiatssoftware ScincePlag (siehe Kapitel 5). Hierzu betrachten wir zunächst die Güte der ausgegebenen Plagiatsstellen und optimieren die Parameter unseres Tools einzeln (Abs. 6.1). In Abschnitt 6.2 wird darauf eingegangen, wie die Ergebnisse mit einem evolutionären Algorithmus verbessert werden können. Abschließend evaluieren wir die Laufzeit und den Speicherbedarf unseres Programms (Abs. 6.3).

6.1 Quantitative Evaluation der gefundenen Textstellen

Dieser Abschnitt beschäftigt sich mit der Frage, wie nützlich die von unserem Tool gefundenen Plagiatsstellen sind. Hierzu muss der Begriff der „Nützlichkeit“ im Bezug auf Plagiatssoftware quantifiziert werden und anhand eines Testdatensatzes berechnet werden (siehe Abs. 6.1.1). Die entsprechenden Definitionen werden in den Abschnitten 6.1.2, 6.1.3 und 6.1.4 hergeleitet. Abschnitt 6.1.5 beschreibt den genauen Versuchsaufbau, gemäß dem wir die Ergebnisgüte von SciencePlag auswerten. In Abschnitt 6.1.6 werden einzelne Parameter variiert. Aus den gewonnenen Daten werden Schlüsse über den Einfluss des jeweiligen Parameters gezogen und solche Werte bestimmt, die sich günstig auf die Ergebnisgüte auswirken. Abschließend ziehen wir ein Fazit über die Möglichkeiten unseres Tools und über den Erfolg der Optimierung individueller Parameter (Abs. 6.1.7).

6.1.1 Vorgehen

Zur Bewertung der Ergebnisse unseres Tools SciencePlag verwenden wir den Text-Alignment-Testkorpus des PAN-Wettbewerbs (siehe auch Abs. 3.2) aus dem Jahr 2013. Dieser Datensatz besteht zum einen aus 3230 Quelldokumenten, sowie zum anderen aus 1827 verdächtigen Dokumenten, bei denen eventuell Textstellen der Quelldokumente plagiiert wurden. Der Testdatensatz beinhaltet zudem eine Liste von vorkommenden Plagiaten, welche im Rahmen des Wettbewerbs als gewünschte Lösung verstanden wird. Es hat sich herausgestellt, dass nicht alle tatsächlich vorliegenden Plagiatsstellen in dieser Lösung vermerkt sind. Die bei der Auswertung auf dem PAN'13-Testkorpus gewonnenen Qualitätsmaße sind somit lediglich eine untere Schranke für die Güte der von unserem Tool gefundenen Ergebnisse. Die Auswertung anhand des PAN'13-Testkorpus dient darüber hinaus der Analyse von Beziehungen zwischen den Parametern unseres Tools (siehe Abs.

5.3) und deren Auswirkung auf das Ergebnis. Hierauf basierend wird eine Optimierung der Parameterwerte durchgeführt.

Als Gütemaße verwenden wir Precision und Recall. Die *Precision* beziffert, wie viele der gefundenen Stellen tatsächlich Plagiatsstellen sind, wohingegen der *Recall* angibt, wie viele der existierenden Plagiatsstellen tatsächlich gefunden wurden. Die Auswertung dieser Gütemaße wird in verschiedene Evaluationsstufen unterteilt, um zu beurteilen, wie exakt Plagiatsstellen eingegrenzt werden können.

6.1.2 Allgemeine Definitionen

Sei R die Menge der von unserem Tool gefundenen Plagiatsstellen. Eine Plagiatsstelle $r \in R$ besteht aus einer Angabe der Textstelle p im verdächtigen Dokument d_p sowie der entsprechenden originalen Textstelle q im Quelldokument d_q. Somit besitzt eine gefundene Plagiatsstelle die Form $r = ((d_p, p), (d_q, q))$. Da sich ein Ergebnis stets auf die Analyse eines einzelnen verdächtigen Dokuments v bezieht, gilt $\forall((d_p, p), (d_q, q)) \in R : d_p = v$. R_D bezeichnet im Folgenden eine vereinfachte Form der Ergebnismenge R, bei der ein Ergebnis $r_D \in R_D$ keine Information zu den genauen Textstellen beinhaltet. Ein Element $r_D = (d_p, d_q)$ besagt also lediglich, dass sich in Dokument d_p Stellen befinden, die aus d_q plagiiert wurden. Da es ein verdächtiges Dokument an mehreren Stellen aus demselben Quelldokument plagiiert haben kann, gilt $|R_D| \leq |R|$.

Die Mengen S und S_D sind analog zu R bzw. R_D definiert (insbesondere gilt auch hier $\forall((d_p, p), (d_q, q)) \in S : d_p = v$), jedoch stammen die Elemente dieser Mengen aus der Lösung des PAN'13-Testkorpus und nicht aus der Ausgabe unseres Tools.

Eine Textstelle ist ein Zahlenintervall $i \subset \mathbb{N}$ im Bereich der von Anfang bis Ende des Dokuments durchnummerierten Zeichen, dargestellt durch dessen Intervallgrenzen $[l, u] \in \mathbb{N}^2$ mit $l = min(i)$ und $u = max(i)$.

6.1.3 Evaluation auf Dokumentenebene

Die erste der verwendeten Evaluationsstufen, genannt „Dokumentenebene", beschäftigt sich lediglich mit den gefundenen Quelldokumenten und ignoriert Angaben zu den genauen Textstellen. Wir betrachten hier also die vereinfachte Ergebnismenge R_D bzw. vereinfachte Lösungsmenge S_D.

Um die Gütemaße Precision und Recall auf Dokumentenebene zu definieren, verwenden wir folgende Funktionen:

$$\begin{aligned} \mathrm{tp}_D(R_D, S_D) &= |R_D \cap S_D| \\ \mathrm{fp}_D(R_D, S_D) &= |R_D \setminus S_D| \\ \mathrm{fn}_D(R_D, S_D) &= |S_D \setminus R_D|. \end{aligned}$$

tp_D stellt hierbei die Anzahl an korrekterweise gefundenen Plagiaten dar (Engl.: *true positive*). Da es sich bei Elementen der Mengen R_D und S_D um Paare aus verdächtigem Dokument und Quelldokument handelt, müssen Elemente im Schnitt dieser beiden Mengen auch in beiden Bestandteilen übereinstimmen. fp_D beschreibt die Anzahl der Ergebnisse unseres Tools, die nicht in der Lösung enthalten sind (Engl.: *false positive*). fn_D wiederum zählt diejenigen Elemente der Lösung, die nicht von unserem Tool gefunden wurden (Engl.: *false negative*).

Mithilfe obiger Funktionen definieren wir Precision und Recall auf Dokumentenebene folgendermaßen:

$$\underset{D}{\text{Precision}}(R_D, S_D) = \begin{cases} \frac{\text{tp}_D(R_D,S_D)}{|R_D|} & \text{falls } |R_D| > 0 \\ 1 & \text{sonst} \end{cases}$$

$$\underset{D}{\text{Recall}}(R_D, S_D) = \begin{cases} \frac{\text{tp}_D(R_D,S_D)}{|S_D|} & \text{falls } |S_D| > 0 \\ 1 & \text{sonst.} \end{cases}$$

Die restlichen verwendeten Evaluationsstufen betrachten nun auch die ausgegebenen Textstellen.

6.1.4 Auswertung auf Textstellenebene

Die Auswertung auf Dokumentenebene sagt nichts darüber aus, wie präzise die Angaben unseres Tools zu den genauen Plagiatsstellen sind. Aus diesem Grund verwenden wir auf Textstellen bezogene Auswertungslevel, die sich lediglich in einer festgelegten minimalen Überlappung von Textstellen unterscheiden.

Der Anteil an Überlappung zweier Textstellen-Intervalle $[l_1, u_1]$ und $[l_2, u_2]$ berechnen wir durch die Funktion $o : \mathbb{N}^4 \to [0, 1] \subset \mathbb{Q}$ mit

$$o([l_1, u_1], [l_2, u_2]) = \begin{cases} \frac{min(u_1,u_2)-max(l_1,l_2)+1}{max(u_1,u_2)-min(l_1,l_2)+1} & \text{falls } max(l_1, l_2) - min(u_1, u_2) \geq 0 \\ 0 & \text{sonst.} \end{cases}$$

Die Überlappung o von zwei Intervallen bestimmt, wie genau die von unserem Tool ausgegebenen Plagiatsstellen den tatsächlichen Plagiatsstellen entsprechen. Zu lang ausgegebene Fundstellen werden dabei in der Evaluation bestraft. Würden wir statt der Überlappung von Textstellen den Anteil an Überdeckung der tatsächlichen Plagiatsstelle verwenden, so könnte stets das gesamte verdächtige Dokument als einzige Textstelle ausgegeben werden und die Überdeckung wäre perfekt.

Wir können nun auswerten, ob zwei Plagiatsstellen $r = (d_{p_r}, p_r), (d_{q_r}, q_r))$ und $s = ((d_{p_s}, p_s), (d_{q_s}, q_s))$ sich gemäß einer gewählten Mindestüberlappung $o_{min} \in [0, 1] \subset \mathbb{R}$ entsprechen. Hierfür wird geprüft, ob die jeweiligen Dokumente übereinstimmen und ob die entsprechenden Textstellen sich mindestens um o_{min} überlappen. Die folgende Funktion $match(r, s, o_{min})$ trifft genau diese Entscheidung:

$$\begin{aligned} &\text{Match}\{((d_{p_r}, p_r), (d_{q_r}, q_r)), ((d_{p_s}, p_s), (d_{q_s}, q_s)), o_{min}\} \\ &= \begin{cases} \top & \text{falls } (d_{p_r} = d_{p_s}) \wedge (d_{q_r} = d_{q_s}) \\ & \qquad \wedge (o(p_r, p_s) \geq o_{min}) \wedge (o(q_r, q_s) \geq o_{min}) \\ \bot & \text{sonst} \end{cases} \end{aligned}$$

Bei der Definition von Precision und Recall auf Textstellenebene ist eine mögliche Uneindeutigkeit zu beachten. Es ist möglich, dass mehrere verschiedene Plagiatsstellen $r_1, r_2, ..., r_n \in R$ in der Ausgabe unseres Tools gemäß $match$ einen Treffer für ein und

dieselbe tatsächliche Plagiatsstelle $s \in S$ darstellen. Um diesen Effekt zu berücksichtigen, definieren wir zunächst folgende Funktionen:

$$\begin{aligned}
\mathrm{tp}(R, S, o_{min}) &= |\{r \in R \,|\, \exists s \in S : \mathrm{Match}(r, s, o_{min})\}| \\
\mathrm{tp}_{\mathrm{unique}}(R, S, o_{min}) &= |\{s \in S \,|\, \exists r \in R : \mathrm{Match}(r, s, o_{min})\}| \\
\mathrm{fp}(R, S, o_{min}) &= |\{r \in R \,|\, \forall s \in S : \neg\, \mathrm{Match}(r, s, o_{min})\}| \\
\mathrm{fn}(R, S, o_{min}) &= |\{s \in S \,|\, \forall r \in R : \neg\, \mathrm{Match}(r, s, o_{min})\}|.
\end{aligned}$$

Die Funktionen tp, fp und fn entsprechen im Wesentlichen ihren zuvor auf Dokumentenebene definierten Gegenstücken, wobei sie nun auf die potenziell ungenaue Auswertung mittels Match verallgemeinert wurden. Das neue Maß $\mathrm{tp}_{\mathrm{unique}}$ zählt die korrekt klassifizierten Plagiatsstellen, jedoch pro Plagiatsstelle in der Lösung maximal eine. Wir definieren nun Precision und Recall auf Textstellenebene als

$$\mathrm{Precision}(R, S, o_{min}) = \begin{cases} \frac{\mathrm{tp}(R,S,o_{min})}{|R|} & \text{falls } |R| > 0 \\ 1 & \text{sonst} \end{cases}$$

$$\mathrm{Recall}(R, S, o_{min}) = \begin{cases} \frac{\mathrm{tp}_{\mathrm{unique}}(R,S,o_{min})}{|S|} & \text{falls } |S| > 0 \\ 1 & \text{sonst.} \end{cases}$$

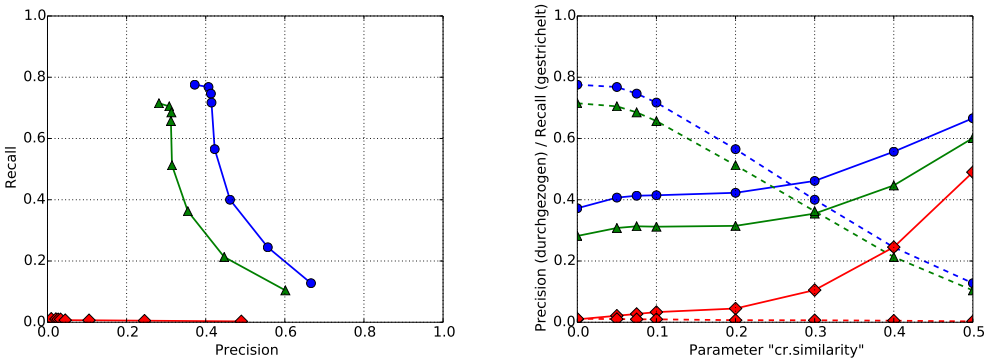
Wie in Abschnitt 6.1.1 beschrieben, soll das Gütemaß Recall die Abdeckung der tatsächlich auffindbaren Plagiatsstellen beschreiben. Um dies zu gewährleisten, haben wir gegenüber der Dokumentenebene das Maß tp_D durch tp_{unique} ersetzt. Würden wir stattdessen tp verwenden, so könnten mehrere leicht versetzte Plagiatsstellen in der Ausgabe als Treffer gelten und würden den Recall zu unseren Gunsten verfälschen. Das Maß Precision berücksichtigt solch mehrfach vorkommenden Ergebnisse nicht, da es als Anteil korrekt klassifizierter Ausgaben verstanden wird. Wir haben uns bei der Auswertung auf die klassischen Gütemaße Precision und Recall beschränkt. Ein drittes Gütemaß, welches die Redundanz in der Ausgabe R unseres Tools bewertet, würde gegebenenfalls noch präzisere Bewertungen gestatten. In der folgenden Evaluation sei $o_{min} \in \{0, \frac{1}{2}\}$.

6.1.5 Versuchsaufbau

Die verdächtigen Dokumente im PAN'13-Testkorpus sind in fünf Kategorien eingeteilt:

1. Keine Plagiate
2. Keine Verschleierung
3. Zufällige Verschleierung
4. Übersetzungsplagiate
5. Zusammenfassende Plagiate

Für die Evaluation unseres Tools SciencePlag verwendet wir die Dokumente in den Kategorien 2 und 3. Kategorie 1 ist für uns von wenig Interesse, da hier dem Namen nach keine Plagiate vorliegen. Übersetzungsplagiate sind mit unserem Tool in aktueller Version nicht auffindbar, daher fällt Kategorie 4 aus der Evaluation heraus. Bei den so genannten zusammenfassenden Plagiaten (Kategorie 5) liegt die Lösung nur auf Dokumentenebene vor, weshalb wir diese Kategorie für unsere Auswertung ebenfalls nicht betrachten. Die



(a) Kategorie 3: Precision-Recall-Kurven **(b)** Kategorie 3: Precision und Recall als Funktion

Abbildung 6.1: Precision- und Recall-Werte bei unterschiedlichen Werten des Parameters `cr.similarity`, PAN'13-Kategorie 3. Blau (Kreise): Dokumentenebene. Grün (Dreiecke): min. Überlappung $o_{min} = 0$. Rot (Rauten): $o_{min} = 0.5$.

verwendeten Kategorien 2 und 3 bestehen aus 170 bzw. 166 verdächtigen Dokumenten. Wenn von einer Auswertung bezüglich einer dieser Kategorien gesprochen wird, so ist gemeint, dass das arithmetische Mittel von Precision und Recall über den verdächtigen Dokumenten aus der jeweiligen Kategorie ermittelt und als Gütemaß genutzt wird.

6.1.6 Auswertung einzelner Parameter

Wir erzeugen nun Versuchsreihen, bei denen jeweils die Werte eines Parameters variiert werden, wobei die anderen Parameter jeweils auf einen voreingestellten, festen Wert gesetzt werden. Die voreingestellten Paramter sind der Zeile „Vorher" in Abbildung 6.8 zu entnehmen. Der Einfluss jedes einzelnen Parameters auf die Güte des Ergebnisses wird anhand von Precision- und Recall-Werten beurteilt. In Abschnitt 6.1.7 werden anschließend die jeweils besten Werte der einzelnen Parameter kombiniert und das Gesamtergebnis bewertet.

Minimale Kosinusähnlichkeit (Candidate Retrieval)

Das Candidate Retrieval-Modul von SciencePlag selektiert eine Vorauswahl an verdächtigen Dokumenten, die anschließend weiter analysiert werden (Abs. 4.4). Es werden dabei sämtliche Dokumente aus der Datenbasis ausgewählt, bei denen die Kosinusähnlichkeit des entsprechenden tf-idf-Vektors mit dem tf-idf-Vektor des verdächtigen Dokuments größer ist als ein gewählter Schwellwert. Dieser Schwellwert der minimalen Kosinus-Ähnlichkeit lautet in unserem Tool SciencePlag `cr.similiarity`.

Abbildung 6.1b zeigt *Precision*- und *Recall*-Werte bezüglich Kategorie 3 des PAN'13-Testkorpus in Abhängigkeit von verschiedenen Werten des Parameters `cr.similiarity`. Die Darstellung zeigt klar den filternden Einfluss des Parameters: Bei höherem Schwellwert gelangen weniger Dokumente durch den Filter, wodurch etwaige Textstellen, die Plagiate der herausgefilterten Dokumente darstellen, unentdeckt bleiben. Der *Recall* sinkt entsprechend. Interessant ist die Steigung der *Precision*-Kurve. Es scheint, dass die Kosinus-Ähnlichkeit der tf-idf-Vektoren in der Tat mit der Wahrscheinlichkeit eines Verhältnisses von Plagiat zu Quelldokument korreliert. Da es sich bei der Darstellung jedoch um Mittelwerte handelt, ist ein vermehrter Einfluss von leeren Ausgabemengen unseres Tools nicht auszuschließen (Precision-Wert von 1). Dieser Einfluss wird an den roten Kurven (Aus-

wertungsstufe $o_{min} = 0.5$) deutlich, denn hier steigt auch bei wenigen richtigen Treffern (*Recall* nahezu 0) der *Precision*-Wert.

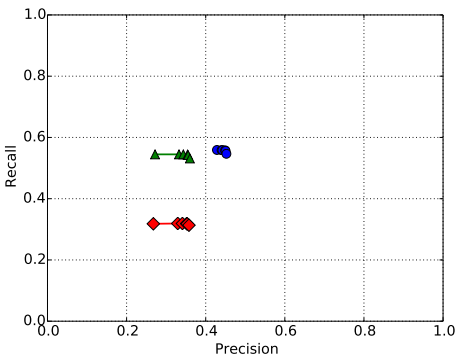
Da das Candidate Retrieval als Vorfilter den *Recall*-Wert nicht bis kaum senken sollte, wird anhand der analysierten Werte ein Schwellwert der Kosinus-Ähnlichkeit von etwa 0.05 empfohlen.

Minimale gemeinsame Präfixlänge (Text Alignment)

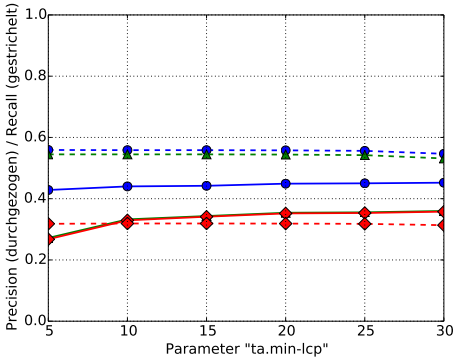
Während des *Textalignment* wird das konstruierte Suffixarray linear auf Übereinstimmungen untersucht (siehe Abschnitt 4.5). Hierzu wird der Parameter `ta.min-lcp` verwendet. Er bestimmt wie viele Worte zweier Textstellen übereinstimmen müssen, damit diese als Fundstelle näher untersucht werden. Abbildung 6.2 zeigt den Verlauf von Precision und Recall für einen Wertebereich von 5 bis 30. Es ist zu erkennen, dass in Kategorie 2 die Wahl des Parameters kaum einen Einfluss auf Precision und Recall hat. Hieraus kann geschlossen werden, dass die Dokumente dieser Kategorie nicht dazu geeignet sind eine Aussage über den Parameter `ta.min-lcp` zu treffen. Kategorie 3 jedoch zeichnet sich durch eine deutliche Änderung der Gütemaße bei der Parameter-Variation aus. Der Verlauf zeigt, dass der Precision-Wert schnell gegen Eins und der Recall-Wert gegen Null tendiert. Hieran ist zu erkennen, dass für hohe Werte von `ta.min-lcp` die Plagiatssuche nur noch eine extrem kleine Auswahl von sehr eindeutigen Plagiaten zurückgibt. Es ist daher anzunehmen, dass ein Parameter-Wert von 5 der Ergebnissuche deutlich zuträglicher ist, als höhere Werte (siehe Abschnitt 6.1.7).

Maximaler Stellenabstand im verdächtigen Dokument (Text Alignment)

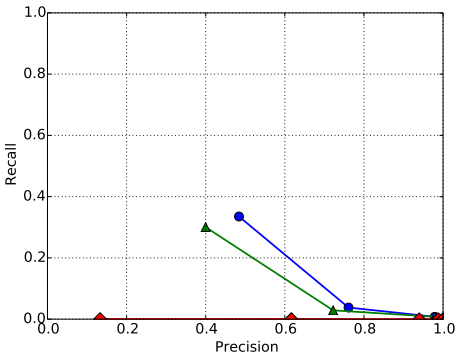
Ein weiterer Bestandteil des *Textalignment* ist das Zusammenfassen von Fundstellen, welche sowohl im verdächtigen Dockument, als auch im Quelldokument dicht beieinander liegen (siehe Abs. 4.5.2). Zu diesem Zweck verwendet das Tool den Parameter `ta.delta-sus`, welcher die maximale Wortdistanz definiert, die zwei entsprechende Plagiate im verdächtigen Dokument voneinander entfernt sein dürfen, sodass sie für eine Zusammenfassung in Frage kommen. Die Wahl dieses Parameters ist insofern relevant, als dass für zu niedrige Werte Fundstellen, in welche lediglich einige Wörter eingefügt wurden, in mehrere erkannte verdächtigen Stellen zerteilt werden. Zu hohe Werte des Parameters `ta.delta-sus` führen jedoch dazu, dass aufeinander folgende Fundstellen, welche als eigenständig betrachtet werden müssen, zusammengefasst werden. Infolge dessen werden Textbereiche zwischen den Fundstellen, welche nicht als Fundstellen erkannt wurden, trotzdem zur Ergebnismenge hinzugefügt werden. Ziel der Auswertung dieses Parameters muss es demnach sein ein Intervall festzulegen, aus dem ein gewählter Wert von `ta.delta-sus` seinen Zweck erfüllt und Randerscheinungen minimiert werden. Die Abbildungen 6.3b und 6.3d zeigen den Verlauf von *Precision* und *Recall* für Parameter-Werte zwischen 0 und 40 für die Kategorien 2 und 3 des Pan'13-Testkorpus. In beiden Fällen ist zu erkennen, dass *Precision* und *Recall* keinen großen Änderungen unterliegen, was darauf hinweist, dass die allgemeine Qualität der Ergebnisse nicht maßgeblich von der Wahl von `ta.delta-sus` abhängig ist. Betrachtet man kleine Änderungen der Gütemaße im Parameterbereich von 0 bis 5, so fällt auf, dass die *Precision* meist leicht abfällt und der *Recall* leicht steigt, oder unverändert bleibt. der Abfall der *Precision* könnte in diesem Fall darauf zurück geführt werden, dass mehr falsche Fundstellen zusammengefasst werden und somit die im *Textalignment* folgende Filterung nach der Länge keine Anwendung mehr findet. Auf diese Weise gelange mehr falsche Fundstellen in das Endresultat und der Anteil an richtigen Fundstellen verringert sich. Der Anstieg des *Recall* kann auf ähnliche Weise gedeutet werden. Nicht nur falsche Fundstellen überstehen durch die Zusammenfassung den Filterprozess, sondern



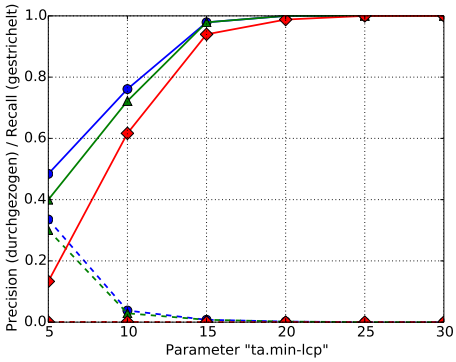
(a) Kategorie 2: Precision-Recall-Kurven



(b) Kategorie 2: Precision und Recall als Funktion

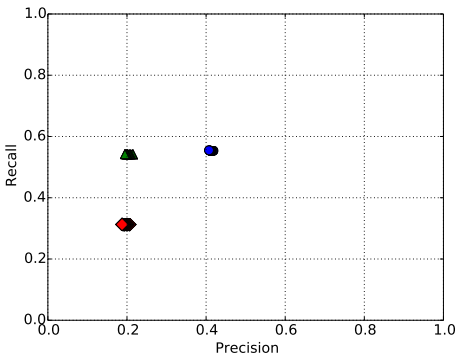


(c) Kategorie 3: Precision-Recall-Kurven

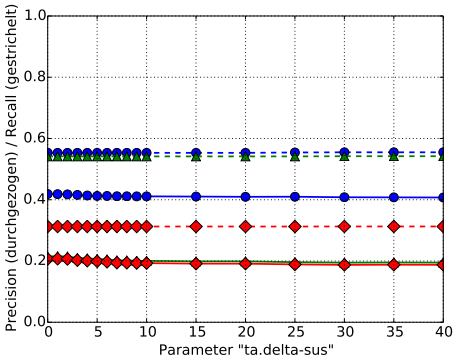


(d) Kategorie 3: Precision und Recall als Funktion

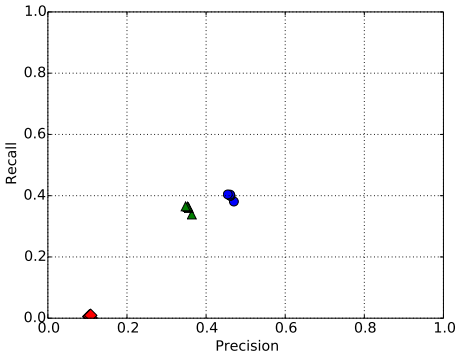
Abbildung 6.2: Precision- und Recall-Werte bei unterschiedlichen Werten des Parameters `ta.min-lcp`, PAN'13-Kategorien 2 und 3. Blau (Kreise): Dokumentenebene. Grün (Dreiecke): min. Überlappung $o_{min} = 0$. Rot (Rauten): $o_{min} = 0.5$.



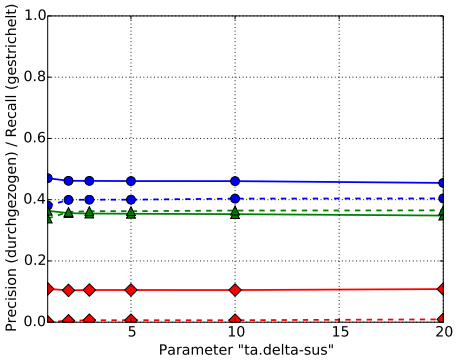
(a) Kategorie 2: Precision-Recall-Kurven



(b) Kategorie 2: Precision und Recall als Funktion



(c) Kategorie 3: Precision-Recall-Kurven



(d) Kategorie 3: Precision und Recall als Funktion

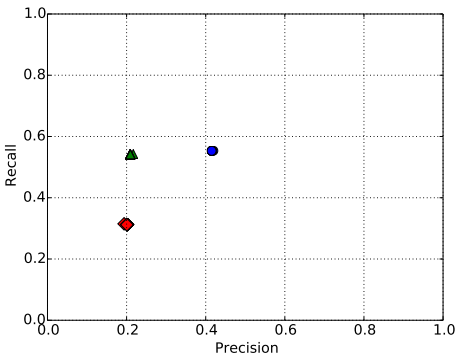
Abbildung 6.3: Precision- und Recall-Werte bei unterschiedlichen Werten des Parameters `ta.delta-sus`, PAN'13-Kategorien 2 und 3. Blau (Kreise): Dokumentenebene. Grün (Dreiecke): min. Überlappung $o_{min} = 0$. Rot (Rauten): $o_{min} = 0.5$.

auch richtige. Hierdurch wird ein größerer Anteil der zu findenden Fundstellen tatsächlich aufgespürt. Betrachtet man die Gegenläufigkeit von *Precision* und *Recall* so scheint es jedoch, dass der Anteil an zusätzlichen falschen Ergebnissen die zusätzlichen richtigen Ergebnisse deutlich überwiegt. Für eine Ergebnismenge mit möglichst wenigen irreführenden Ergebnissen sollte demnach ein Wert nahe 0 für den Parameter `ta.delta-sus` vorgezogen werden.

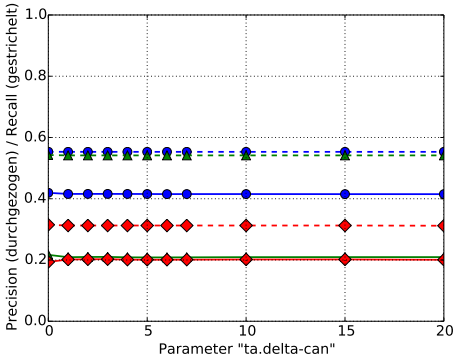
Maximaler Stellenabstand im Kandidatendokument (Text Alignment)

Analog zu `ta.delta-sus` wird mittels des Parameters `ta.delta-can` der maximale Abstand zweier Fundstellen im Kandidatendokument definiert, welcher eine Zusammenfassung während des *Textalignment* erlaubt. Die Abbildungen 6.4b und 6.4d zeigen den Verlauf der Gütemaße bei Variation dieses Parameters.

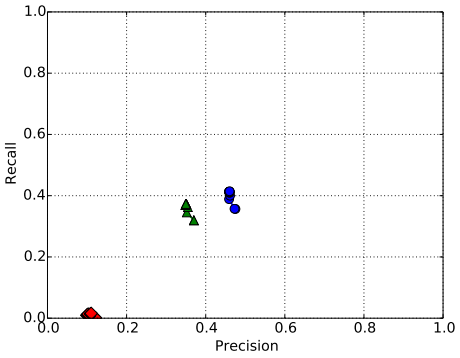
Im Gesamtbild ist zu erkennen, dass `ta.delta-can` die Gütemaße, vor allem in Kategorie 3, stärker beeinflusst als `ta.delta-sus`. Wieder entstehen Änderungen jedoch hauptsächlich bei kleinen Parameterwerten unter 5. Analog zu `ta.delta-sus` sinkt die *Precision* und der *Recall* steigt. Es ist anzunehmen, dass dies auf dieselbe Weise erklärt werden kann, wie oben beschrieben. Gleichbleibend kann gesagt werden, dass vor allem kleine Parameterwerte Relevanz bezüglich der Gütemaße besitzen und ein möglichst kleiner Parameterwert einem Zuwachs an falschen Ergebnissen entgegenwirkt. Das Intervall



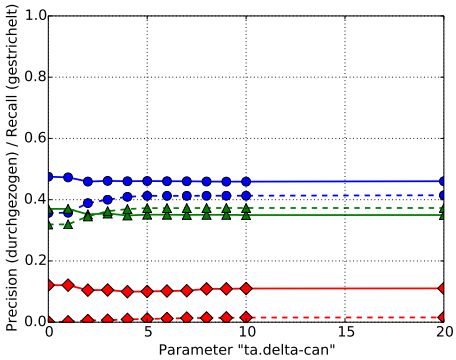
(a) Kategorie 2: Precision-Recall-Kurven



(b) Kategorie 2: Precision und Recall als Funktion



(c) Kategorie 3: Precision-Recall-Kurven



(d) Kategorie 3: Precision und Recall als Funktion

Abbildung 6.4: Precision- und Recall-Werte bei unterschiedlichen Werten des Parameters `ta.delta-can`, PAN'13-Kategorien 2 und 3. Blau (Kreise): Dokumentenebene. Grün (Dreiecke): min. Überlappung $o_{min} = 0$. Rot (Rauten): $o_{min} = 0.5$.

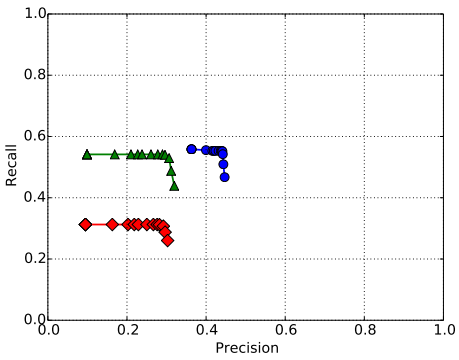
in dem dieser Parameter merkliche Auswirkungen auf das Ergebnis hat ist jedoch etwa doppelt so groß wie beim Parameter `ta.delta-sus` und liegt etwa bei 0 bis 5. Man kann sagen, dass die Erhöhung dieses Parameters als sinnvollere Möglichkeit genutzt werden kann, um den *Recall* auf den gegebenen Testdaten zu verbessern.

Minimale Plagiatslänge (Text Alignment)

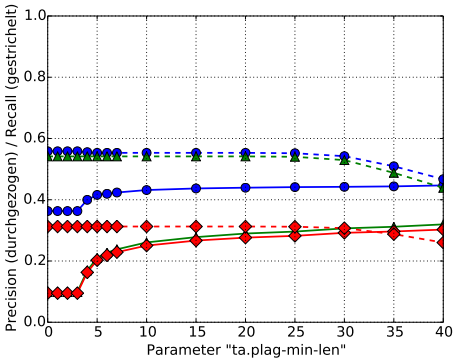
Die minimale Plagiatslänge (`ta.plag_min_len`) gibt an, wie viele Wörter eine Plagiatsstelle mindestens umfassen soll. Beispielsweise sind alle Plagiatsstellen bei einem Wert von 5 mindestens fünf Wörter lang. Dieser Parameter kommt in der Nachbearbeitung zur Anwendung, damit keine Passagen wegfallen, die durch Merge (siehe Absatz 4.5.2) entstanden sind.

Bei Plagiaten der Kategorie 2 im PAN'13-Testkorpus (Abbildung 6.5b) steigt die Precision mit steigendem Wert der minimalen Plagiatslänge bis zum Wert 10. Danach flacht die Kurve ab. Bei sehr hohen Werten der minimalen Plagiatslänge, circa 35, fällt dagegen der Recall-Wert ab. Deswegen ist es bei Plagiaten ohne Verschleierung vermutlich am Besten, einen Wert von 15-20 zu wählen.

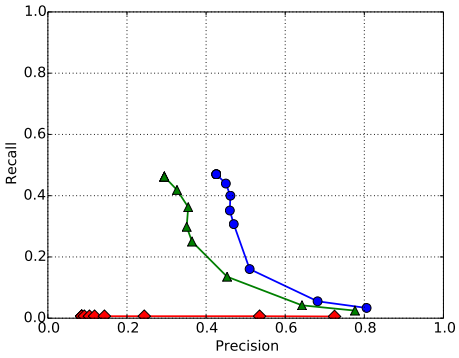
In der Kategorie 3 im PAN'13-Testkorpus (Abbildung 6.5d) steigt die Precision stetig, je höher der Parameter der minimalen Plagiatslänge gewählt wird. Dagegen fällt der Recall-Wert ab, wenn der Parameter größer als 3 gewählt wird. Der beste Wert für Pla-



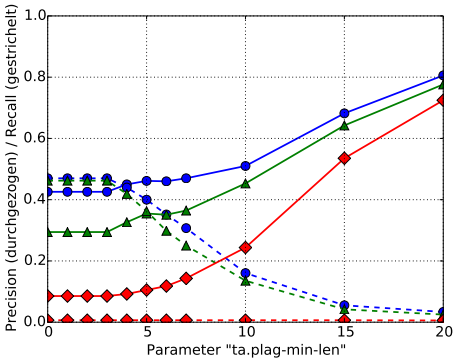
(a) Kategorie 2: Precision-Recall-Kurven



(b) Kategorie 2: Precision und Recall als Funktion



(c) Kategorie 3: Precision-Recall-Kurven



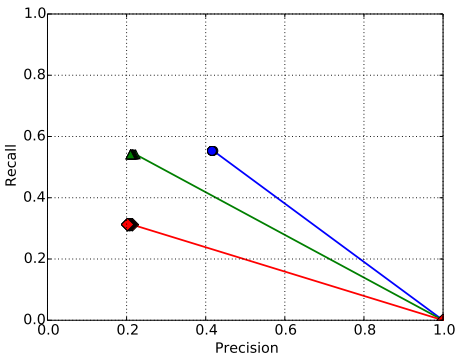
(d) Kategorie 3: Precision und Recall als Funktion

Abbildung 6.5: Precision- und Recall-Werte bei unterschiedlichen Werten des Parameters `ta.plag-min-len`, PAN'13-Kategorien 2 und 3. Blau (Kreise): Dokumentenebene. Grün (Dreiecke): min. Überlappung $o_{min} = 0$. Rot (Rauten): $o_{min} = 0.5$.

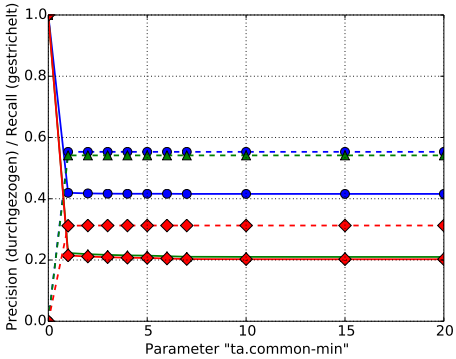
giate mit Verschleierung liegt deshalb zwischen 4 und 5, wo sich die Kurven des Precision und Recall für die Dokumentenebene und einer minimalen Überlappung von 0 kreuzen. Insgesamt ist entweder ein Wert zwischen 5 und 10 empfehlenswert, wenn mit einer Suche beide Arten von Plagiaten aufgedeckt werden sollen, oder getrennte Suchläufe mit eigenen Parametern für die beiden Kategorien.

Mindestanzahl häufiger Formulierungen (Text Alignment)

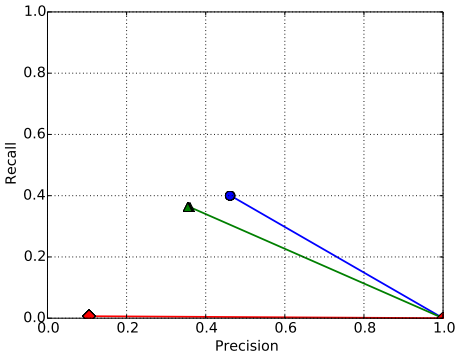
Im Postprocessing versucht SciencePlag häufig auftretende Formulierungen herauszufiltern, da diese nicht als Plagiat anzusehen sind. Dafür werden wie in Abschnitt 4.5.2 beschrieben, Fundstellen als domänenspezifische Formulierung gewertet, wenn diese in einer Mindestanzahl von Dokumenten vorkommen.
Die Abbildung 6.6b zeigt die *Precision*- und *Recall*-Werte für die Variation des Parameters `ta.common-min` auf verdächtigen Dokumenten der Kategorie 2. Abbildung 6.6d wiederholt dies für Kategorie 3. Für beide Kategorien fällt auf, dass die Varianz des `ta.common-min`-Parameters nur einen minimalen Einfluss auf Precision und Recall hat. Als Ausnahme ist dabei ein Wert unter 1 zu betrachten, bei dem sämtliche Fundstellen als häufige Formulierung markiert und somit entfernt werden. Durch die fehlende Varianz lässt sich kein optimaler Wert für den `ta.common-min`-Parameter bestimmen, zumindest nicht mit den vorliegenden Testdaten. Allerdings ist festzustellen, dass durch einen niedrigen Wert für



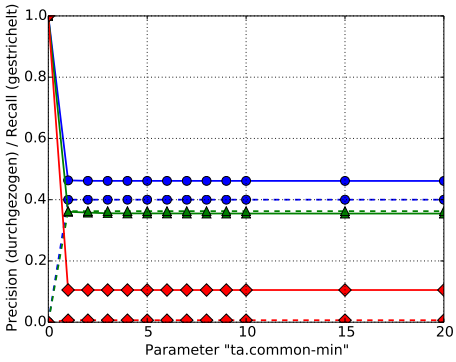
(a) Kategorie 2: Precision-Recall-Kurven



(b) Kategorie 2: Precision und Recall als Funktion



(c) Kategorie 3: Precision-Recall-Kurven



(d) Kategorie 3: Precision und Recall als Funktion

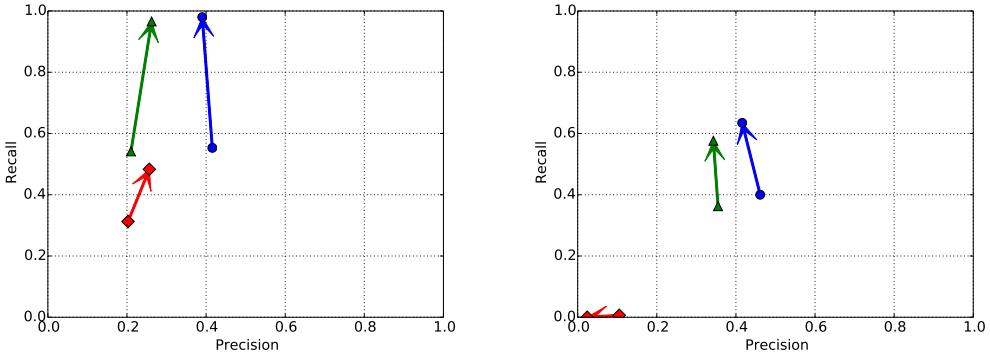
Abbildung 6.6: Precision- und Recall-Werte bei unterschiedlichen Werten des Parameters `ta.common-min`, PAN'13-Kategorien 2 und 3. Blau (Kreise): Dokumentenebene. Grün (Dreiecke): min. Überlappung $o_{min} = 0$. Rot (Rauten): $o_{min} = 0.5$.

`ta.common-min` korrekt gefundene Fundstellen entfernt werden. Mit einem hohen Wert werden lediglich Ergebnisse zusätzlich angezeigt, die möglicherweise nicht als Plagiat anzusehen sind, aber dennoch kopierte Textstellen oder Paraphrasen darstellen.

6.1.7 Vergleich von optimierten und voreingestellten Parametern

Im Folgenden vergleichen wir die Precision- und Recall-Werte unseres Tools zwischen den Standardeinstellung der Parameter und einer Parameterkombination, die aus den in Abs. 6.1.6 selektierten Werten besteht. Die genauen Werte der Parameter für diesen Vergleich finden sich in Abbildung 6.8. Die Veränderung von Precision und Recall von der Standardeinstellung hin zu selektierten Parametern ist durch Pfeile in Abbildung 6.7 dargestellt. Im Allgemeinen ist eine deutliche Verbesserung der Ergebnisgüte sichtbar, wenn wir statt den Voreinstellungen die gemäß unserer Auswertung gewählten Parameter verwenden. Eine detailliertere Analyse der Ergebnisse wird im Weiteren beschrieben.

Es fällt positiv auf, dass wir in der Kategorie 2 („keine Verschleierung") einen Recall-Wert von nahezu 100% sowohl auf Dokumentenebene als auch auf Ebene einzelner Textstellen erreichen, solange die minimale Überlappung o_{min} zwischen unserem Fund und der Lösung des PAN'13-Testkorpus uninteressant ist (vgl. blaue und rote Pfeile in Abb. 6.7a). Der Precision Wert hingegen erreicht hier keine 50% und nimmt ab, wenn einzelne Textstellen anstelle von ganzen Dokumenten betrachtet werden. Aufgrund der zuvor



(a) Kategorie 2 (Keine Verschleierung) **(b)** Kategorie 3 (Zufällige Verschleierung)

Abbildung 6.7: Precision- und Recall-Werte mit Standardeinstellungen (Anfang des Pfeils, vgl. Tab. 6.8, „Vorher") sowie Kombination der laut unserer Auswertung günstigen Parameter (Ende des Pfeils vgl. Tab. 6.8, „Nachher"). Blau (Kreise): Dokumentenebene. Grün (Dreiecke): min. Überlappung $o_{min} = 0$. Rot (Rauten): $o_{min} = 0.5$.

beschriebenen Unvollständigkeit der Lösung im PAN'13-Korpus sind die Precision-Werte schwer zu bewerten.

Auffällig ist, dass der Recall-Wert in Kategorie 2 auch mit optimierten Parametern auf ca. 50% sinkt, wenn eine minimale Überlappung von $o_{min} = 0.5$ gefordert wird. Dieses Resultat lässt zwei mögliche Schlüsse zu. Zum einen kann es sein, dass kleinere Teilabschnitte größerer Plagiate als einzelne Fundstellen ausgegeben werden und somit nicht die minimale Überlappung mit der Lösung erreichen. Zum anderen kann es sein, dass unser Tool die Definition eines Plagiats großzügiger definiert als die Lösung des PAN'13-Testkorpus. Wenn im Schritt Merge des *Textalignment* (siehe Abs. 4.5.2) zu weit auseinander liegende Abschnitte zusammengefasst werden, so würde dies zu einer größeren Plagiatsstelle führen, mit entsprechend sinkender Überlappung.

Die Auswertung anhand von Kategorie 3 („zufällige Verschleierung") zeigt, dass die dort vorkommende Verschleierung eine Schwierigkeit für unser Tool darstellt. Selbst auf Dokumentenebene erreichen wir hier keine 70% an Recall (Abb. 6.7a). Die Precision-Werte auf Dokumentenebene und Textstellenebene ohne Mindestüberlappung sind hingegen höher als in der Kategorie 2 („keine Verschleierung"). Auffällig sind die sehr schlechten Werte für eine minimale Überlappung von $o_{min} = 0.5$: Der Recall ist nahe Null und die Precision sinkt, wenn die Parameter gemäß unserer Auswertung optimiert werden. Der Grund für die sinkende Precision ist der geringer werdende Einfluss von leeren Ergebnismengen, die laut Definition den Precision-Wert Eins besitzen.

Zusammenfassend lässt sich sagen, dass die Optimierung der Parameter unseres Tools SciencePlag erfolgreich war. Als Ergebnis erhalten wir einen sehr guten Recall bei Copy-Paste-Plagiaten (Kategorie 2), wobei sich über den Precision-Wert aufgrund der mangelnden Vollständigkeit der vorhandenen Lösung wenig aussagen lässt. Die Auswertung zeigt Schwierigkeiten unseres Tools mit verschleierten Plagiaten sowie eine geringe Überdeckung der gefundenen Textstellen mit denen der Lösung. Zukünftige Verbesserungen der Ergebnisgüte sollten sich daher auf die letztgenannten Aspekte konzentrieren.

	cr. similarity	ta. min-lcp	ta. delta-sus	ta. delta-can	ta. common-min	ta. plag-min-len
Vorher	0.3	3	3	3	10	5
Nachher	0.05	5	0	5	1	5

Abbildung 6.8: Konfigurationen der Parameter unseres Tools. „Vorher" entspricht der Standardeinstellung, „Nachher" wurde gemäß der einzelnen Auswertungen in diesem Kapitel gewählt.

6.2 Erweiterte Optimierung der Parameterauswahl

Die soweit beschriebenen Analysen von SciencePlag beruhen auf Annahmen zu möglichen guten Parameterkombinationen, die durch Betrachtung der Algorithmen gerechtfertigt erscheinen. Dabei wurde jedoch nur Teilbereiche und Ausschnitte des gesamt möglichen Parameterraums genutzt. Durch gezielte Experimente mit SciencePlag stellten wir fest, dass es weitere Parameterkombination und Bereiche des Parameterraums gibt, welche zu guten Ergebnissen führen.

Eine vollständige Suche nach einer möglichst guten Auswahl von Parametern ist jedoch sehr aufwändig. Unter der Annahme, dass die Werte der Parameter für die Kosinusähnlichkeit (`cr.similarity`) und die minimale Plagiatslänge (`ta.plag_min_len`), sowie der Schwellwert für domänenspezifische Formulierungen (`ta.common_min`) konstant gewählt werden können, gilt es noch eine möglichst gute Kombination für die Parameter `ta.min_lcp`, `ta.delta_sus` und `ta-delta_can` zu finden. Diese Annahme ist durch die in Abschnitt 6.1 gemachten Beobachtungen gerechtfertigt.

Setzen wir den Suchbereich der verbleibenden Parameter jeweils fest auf den Wertebereich $[3, 50]$, wären somit insgesamt 48^3 Kombinationen zu testen. Bei einer Laufzeit von jeweils ca. einer Minute ($T = 1$) für den gesamten PAN-Datensatz, erreichen wir damit eine Gesamtlaufzeit von 110592 Minuten oder ~ 77 Tagen. Aus diesem Grund haben wir uns entschieden, den Suchraum mit Hilfe eines *Evolutionären Algorithmus (EA)* zu erkunden. Dieser nutzt eine Population von Parameterkombinationen und bewertet jede Parameterkombination anhand einer Fitnessfunktion. Die beste bisherige Lösung wird in die nächste Generation übernommen, somit wird eine monotone Verbesserung der Lösung sichergestellt. Zudem werden die besten k Lösungen zur Rekombination genutzt, wobei zufällige Mutationen einzelner Parameterwerte möglich sind. Dabei unterliegt die Mutationspannweite den Rekombinationen und der Mutation selbst. Wir haben folgende parametrisierbare Fitnessfunktion verwendet:

$$f_\alpha(recall, precision) := ((recall + 1) \cdot \alpha)^2 + ((precision + 1) \cdot (1 - \alpha))^2$$

Durch den Parameter α lässt sich der Tradeoff zwischen Recall und Precision justieren, wobei sich $\alpha = 0.75$ gut bewährt hat. Algorithmus 5 beschreibt das Vorgehen des EA zur Parametersuche. Pro Runde wird SciencePlag mit jeder Parameterkombination p auf den PAN-Dokumenten d aufgerufen. Danach wird der aktuelle Lauf ausgewertet und Recall und Precision berechnet und damit die Fitnessfunktion ausgewertet. Die Fitnesswerte werden mit den zugehörigen Parameterkombination werden zwischenspeichert. Nachdem alle Parameterkombination abgearbeitet sind, wird das beste Ergebnis in die nächste Runde übernommen. Die restlichen Parameterkombinationen werden rekombiniert und mutiert.

Die Laufzeit des EA ist bestimmt durch die Anzahl der Runden r und Größe der Population $p = |P|$. Sie beträgt asymptotisch $\mathcal{O}(rp) \cdot T$, wobei T die Laufzeit der Evaluation selbst bezeichnet. Bei einer Wahl von $r = p = 20$ und $T = 1$, was einer Minute entspricht, ergibt dies 400 Minuten Laufzeit. Durch eine Reduzierung der Anzahl genutzter Dokumente konnte die Laufzeit weiter reduziert werden. Für die Suche nach optimalen Pa-

EA zur Parametersuche
Eingabe: Anzahl der Runden r
Eingabe: PAN Dokumente d
Ausgabe: Erfolgreichste Parameterkombination
1: Initialisiere zufällige Population von Parametervektoren P
2: **for** $1..r$ **do**
3: Initialisiere Array $fitness \leftarrow []$ ▷ Beinhalted Fitness-Werte
4: **for** $p \in P$ **do**
5: Rufe SciencePlag mit Parametern p auf Dokumenten d auf
6: Werte $recall$ und $precision$ aus ▷ Evaluation
7: $fitness[p] \leftarrow f_\alpha(recall, precision)$
8: **end for**
9: Sortiere $fitness$ absteigend
10: Initialisiere P_{neu}
11: $P_{neu}[1] \leftarrow fitness[1].p$
12: Rekombiniere und mutiere P und übernehme in P_{neu}
13: $P \leftarrow P_{neu}$
14: **end for**
15: Gebe $P[1]$ zurück

Algorithmus 5: Parameteroptimierung EA.

rametern wurden aus den Kategorien 2 und 3 Stichproben der Größe 50 gezogen und der EA mit den beiden Datensätzen zusammen ausgeführt. Diese Vorgehen wurde 8 mal wiederholt. Abbildungen 6.9 und 6.10 geben einen Überblick über die Ergebnisse. Es wurden jeweils die besten Parameterkombinationen ausgewählt und SciencePlag mit diesen dem gesamten PAN-Datensatz (Kategorie 2 und 3) ausgeführt. Die Ergebnisse für Recall und Precision sind als Boxplots dargestellt. Das arithmetische Mittel ist jeweils durch den roten Quadrat angezeigt. Die einzelnen Parameterkombinationen sind auf der X-Achse als Vektor in folgender Reihenfolge aufgetragen: (`cr.similarity`, `ta.min_lcp`, `ta.delta_sus`, `ta.delta_can`, `ta.common_min`, `ta.min_len`).

Als Ergebnis lässt sich feststellen, dass ein niedriger `ta.min_lcp` Wert zwischen 2 und 5 und höhere `ta.delta_sus` und `ta.delta_can` Werte zwischen 20 und bis gar 40 auch zu guten Ergebnissen führen. Beim ersten Boxplot fällt auf, dass ein niedriger `ta.min_lcp` Wert von 2 zu einem Overfitting führen kann, was bedeutet, dass sehr gute Recall-Werte erreicht werden, die Precision jedoch stark abnimmt. Ansonsten erhält man guten Recall und einigermaßen gute Precision in allen Evaluationsebenen. Das Recall-Precision-Verhältnis wird auch durch die Wahl des Parameters α der Fitnessfunktion beeinflusst.

Zu berücksichtigen ist die zufallsbasierte Natur des Suchverfahrens, welche kein globales Optimum garantiert. Die Ergebnisse sind i.d.R. lediglich lokale Optima der Fitnessfunktion. Eine abgeänderte Fitnessfunktion, bzw. mit geändertem α, könnte zudem eine andere Struktur aufweisen und somit zu abweichenden. Bei einer Reihe von Versuchen mit verschiedenen Werten für α ($0.25, 0.5, 0.6, 0.7, 0.75, 0.9$ und 1.0), stellten wir jedoch die besten Resultate für $\alpha = 0.75$ fest.

6.3 Laufzeit und Speicherverbrauch

Im folgenden Abschnitt sollen die Auswirkungen verschiedener Eingabeparameter auf die Laufzeit und den Speicherverbrauch unseres Tools untersucht werden. Zu diesem Zweck

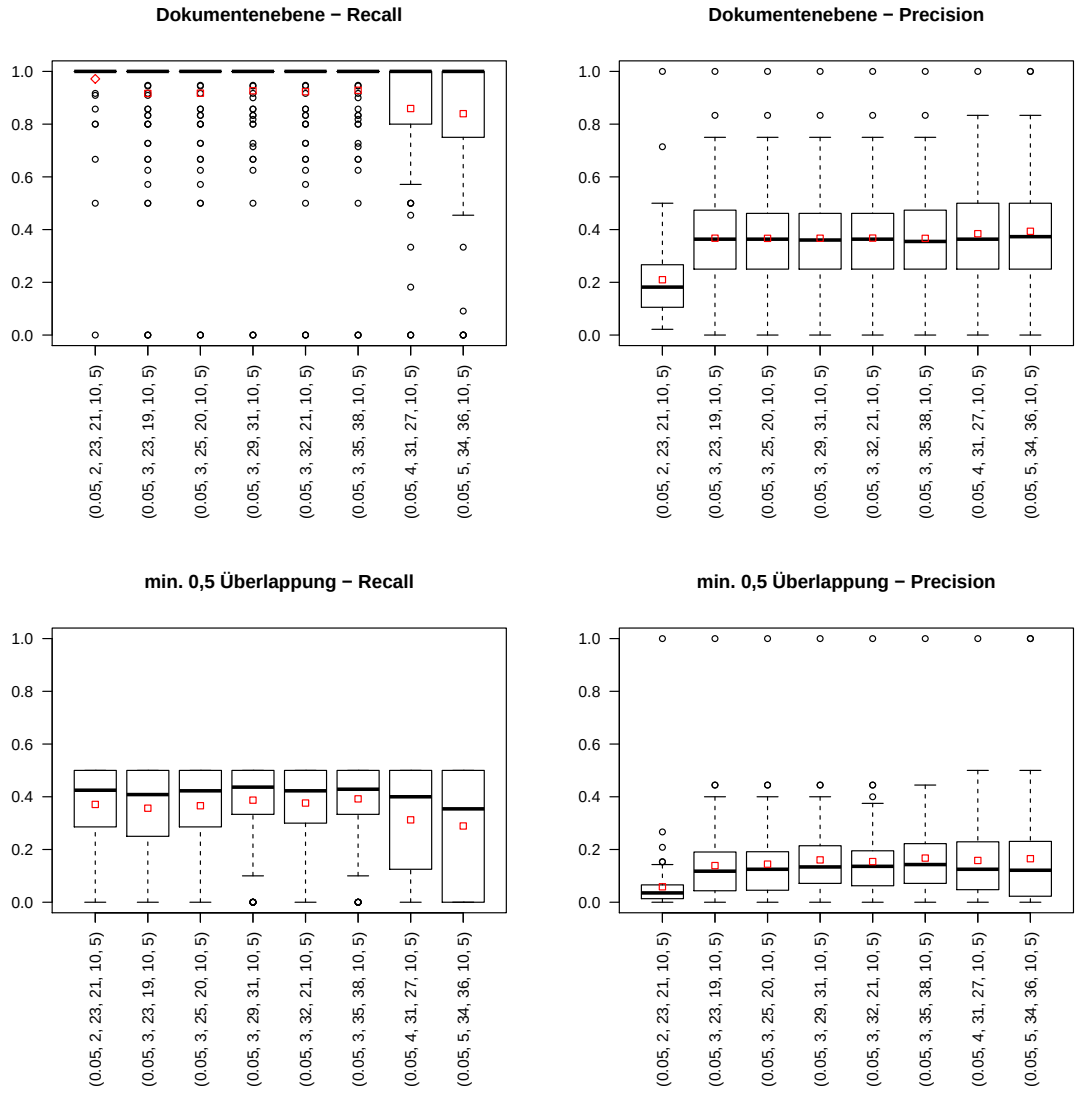


Abbildung 6.9: Boxplots für die einzelnen Evaluationsebenen: Dokumentenebene und 0.5-Überlappung. Links sind die Boxplots der Recall Werte, rechts die der Precision Werte. Man sieht, dass auf der Dokumentenebene für die gefundenen Parameterkombinationen im Mittel sehr gute Recall-Werte erreicht werden. Die Precision-Werte fallen schlechter aus. Für eine Überlappung von min. $o_{min} = 0.5$ fallen die Ergebnisse etwas weniger gut aus, es werden hier Recall-Werte von max. 0.5 erreicht und die arithmetischen Mittel sind dementsprechend gesunken. Auch die Precision-Werte fallen hier weniger gut aus als für auf der Dokumentenebene.

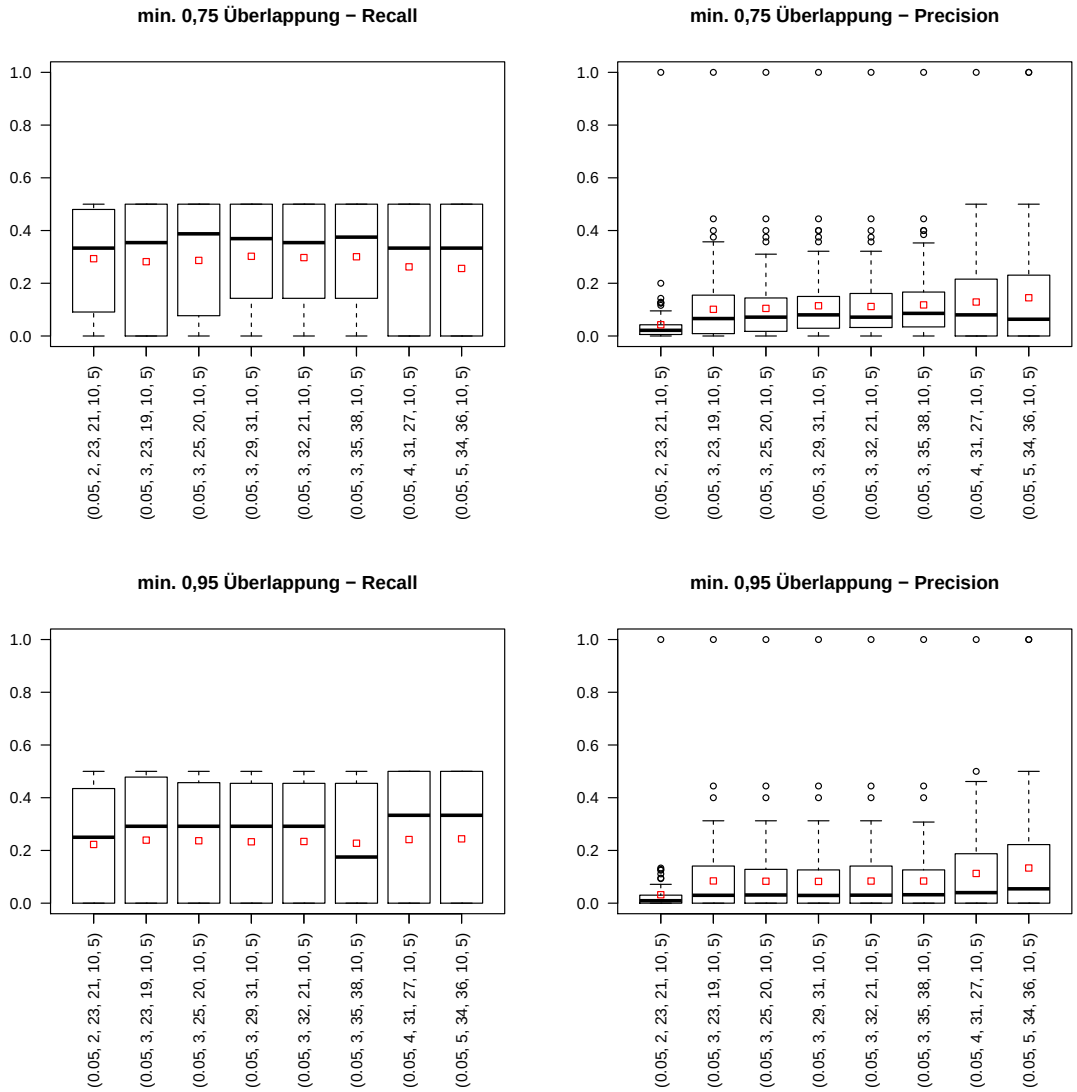


Abbildung 6.10: Boxplots für die einzelnen Evaluationsebenen: 0.75-Überlappung und 0.95-Überlappung. Links sind die Boxplots der Recall Werte, rechts die der Precision Werte. Für die Überlappungen von min. $o_{min} = 0.75$ und $o_{min} = 0.95$ fallen die Recall- und Precision-Werte nicht wesentlich schlechter aus als für die min $o_{min} = 0.5$ Überlappung (s. o.). Man sieht dass die Verteilung der Ergebnisse den Bereich der Recall-Werte zwischen $0 - 0.5$ ausfüllt, da die unteren Quartile in den meisten Fällen bei 0 liegen.

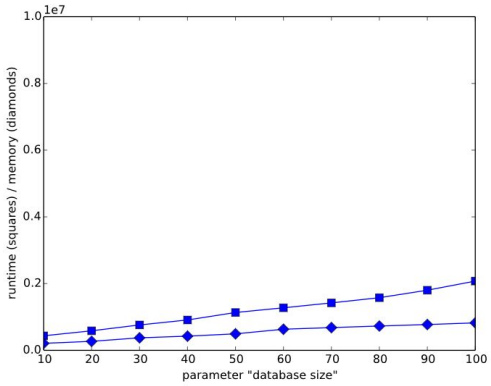


Abbildung 6.11: Die Entwicklung der durchschnittlichen Laufzeit (Quadrate) und des Speicherbedarfs (Rauten) von SciencePlag bei steigender Größe der Dokumentenbasis. Angaben der Datenbankgröße in Prozent, anteilig an der Größe des PAN'13-Testkorpus. Angabe der Laufzeit in Mikrosekunden und des Speicherbedarfs in 100 Bits.

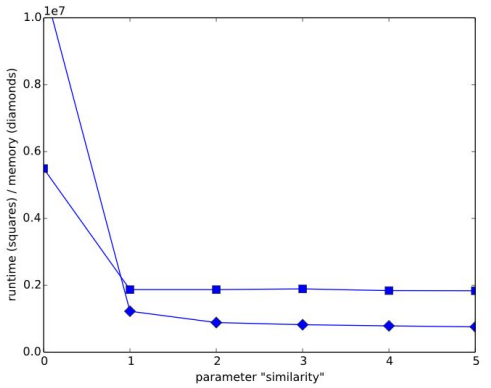


Abbildung 6.12: Die Entwicklung der durchschnittlichen Laufzeit (Quadrate) und des Speicherbedarfs (Rauten) von SciencePlag bei steigender Größe der *Similarity*. Angaben der *Similarity* in Zehnteln. Angabe der Laufzeit in Mikrosekunden und des Speicherbedarfs in 100 Bits.

wird der Bedarf an Speicher mithilfe der C++-Bibliothek *malloc_count*[1] festgestellt. Die Laufzeit des Tools wird durch simple Systembefehle im Code ermittelt, welche die Laufzeit messen, diese jedoch selbst nicht beeinträchtigen. Als Eingabeparameter wird die Größe der verwendeten Dokumentenbasis, sowie die im vorherigen Abschnitt evaluierten Systemparameter verwendet. Zum Zweck der Eingabeparameter-Variation werden alle Eingabeparameter auf Standardwerten festgehalten und lediglich ein Parameter in seiner Größe variiert. Wir untersuchen lediglich Aufrufe des Tools mit der Option *find*, da dies der rechenaufwändigste Befehl ist, der zudem implizit Aufrufe von *retrieve* und *align* beinhaltet.

Auswertung anhand der Dokumentenbasis-Größe

Im Folgenden wird die Laufzeit und der Speicherverbrauch von SciencePlag in Abhängigkeit von variierenden Größen der Dokumentenbasis analysiert. Hierzu wird der PAN-Testkorpus in zehn verschieden große Dokumentmengen aufgeteilt, bei denen jede ein Zehntel des Gesamtkorpus größer ist, als die vorhergehende. Die zu untersuchenden Dokumente aus der Liste *random-obfuscation* (siehe Abschnitt 6.1.1) werden anhand der Stan-

[1] `https://panthema.net/2013/malloc_count/` (Zuletzt abgerufen: 06.04.2016)

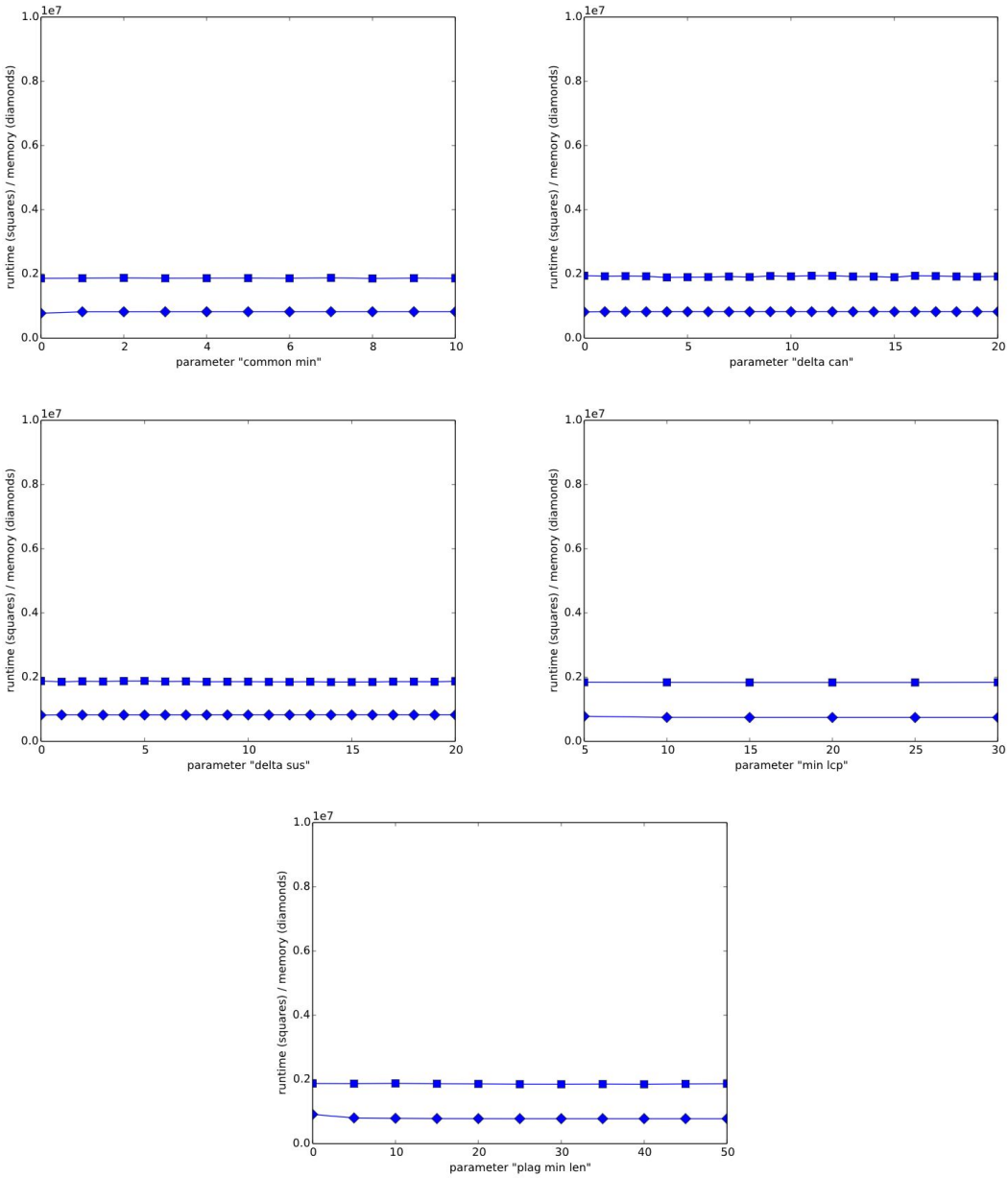


Abbildung 6.13: Die Entwicklung der durchschnittlichen Laufzeit (Quadrate) und des Speicherbedarfs (Rauten) von SciencePlag bei steigender Größe der System-internen Parameter des Text Alignments. Angabe der Laufzeit in Mikrosekunden und des Speicherbedarfs in 100 Bits.

dardwerte aller system-internen Parameter von SciencePlag auf allen zehn Teilmengen auf Plagiate untersucht. Die Ergebnisse der Analysen sind in Abbildung 6.11 dargestellt. Zu erkennen ist, dass sowohl Laufzeit als auch Speicherbedarf, für Datenbasis-Größen im Bereich von einigen Hundert bis einigen Tausend Dokumenten, linear ansteigen. Die linearen Verfahren zur Suche beim *Candidate-Retrieval* zur Erkennung ähnlicher Dokumente, sowie im *Text-Alignment* zur Erkennung von Plagiaten, arbeiten also wie erwartet.

Wir stellen somit fest, dass unser Tool dazu geeignet ist, große Datenmengen als Grundlage zur Plagiatserkennung zu nutzen, ohne unerwartet Grenzen bei der Handhabbarkeit aufzuzeigen.

Auswertung anhand der System-internen Parameter

Nun wird die Laufzeit und der Speicherverbrauch von SciencePlag anhand variierender Größen der system-internen Parameter analysiert. Die zu variierenden Parameter sind analog zu den Parametern in Abschnitt 6.1.1 gewählt. Die Test-Datenbank besteht aus dem vollständigen PAN-Testkorpus. Laufzeit und Speicherbedarf werden als Durchschnitt über allen Überprüfungen der zu untersuchenden Dokumente aus der Liste *random-obfuscation* (siehe Abschnitt 6.1.1) berechnet. Die Ergebnisse sind in den Abbildungen 6.12 und 6.13 dargestellt. Deutlich zu erkennen ist, dass die Wahl der Parameter in den getesteten Wertebereichen keinen merklichen Einfluss auf Laufzeit und Speicherbedarf hat. Eine Ausnahme hierzu bildet der Parameter *Similarity*, wie die Abbildung 6.12 zeigt. Hier ist für den Wert `cr.similarity` $= 0$ zu erkennen, dass Laufzeit und Speicherbedarf einen klaren Ausreißer besitzen. Die Laufzeit steigt auf das beinahe dreifache der sonstigen Werte und der Speicherbedarf erzielt sogar ein Zehnfaches des normalen Wertes. Bei genauerer Analyse des Algorithmus von SciencePlag ist eindeutig zu erkennen, woher diese Ausreißer rühren. Wird das geforderte Ähnlichkeitsmaß für das *Candidate-Retrieval* auf Null gesetzt, werden sämtliche Dokumente aus der Dokumentenbasis an das *Text-Alignment* zur genaueren Untersuchung weitergegeben. Dies führt dazu, dass aus allen Dokumenten ein Suffixarray erstellt wird und dieses vollständig nach Plagiaten durchsucht wird. Hierdurch verliert das Tool jeglichen Vorteil, den es durch das *Candidate-Retrieval* erhalten sollte. Abseits dieses Ausreißers ist das Systemverhalten jedoch als linear (Datenbank-Größe), bzw. konstant (übrige Parameter) in Laufzeit und Speicherbedarf anzusehen.

KAPITEL 7

Zusammenfassung und Ausblick

In diesem Kapitel werden die erreichten Ziele und Ergebnisse betrachtet. Es wird rückblickend der Ablauf der PG SciencePlag beleuchtet. Abschließend wird ein Ausblick auf die möglichen Fortführungen der Arbeiten der PG gegeben.

7.1 Ergebnisse

Wie bereits in der Einleitung (siehe Kapitel 1) dargelegt wurde, war das Ziel der Projektgruppe die Konzeption und Umsetzung einer Software zur effizienten Plagiatserkennung [5]. Dabei sollten Methoden der Stringology und des Information Retrieval zum Einsatz kommen. Textuelle Plagiatserkennung stand somit im Fokus. Die von uns gemeinsam entwickelte Software sollte im besten Falle in der Lage sein neben *Copy&Paste* Plagiate, auch aktiv verschleierte Plagiate zu entdecken. Anregung war, neben den immer wieder in den Medien auftauchenden Plagiatsvorfällen prominenter Persönlichkeiten, der PAN-Wettbewerb, welcher im Rahmen der *CLEF-Konferenz* stattfindet. Unsere Software sollte den Vergleich zu den Teilnehmern der letzten Jahre nicht scheuen müssen. Somit wurde unsere Software SciencePlag mit dem Ziel im PAN-Wettbewerb konkurrenzfähig zu sein entwickelt.

Jedoch existierten unterschiedliche Anforderungen zwischen PAN-Wettbewerb und dem Ziel der PG, da die Teilnehmer des PAN-Wettbewerbs in zwei Kategorien eingeteilt werden. Es wird dort zwischen der Aufgabe des Text Alignment und der Suche nach möglichen Kandidaten mit Hilfe einer gegebenen Suchmaschine unterschieden (siehe Kapitel 3). Diese Unterscheidung haben wir nicht getroffen, sondern wir haben mit SciencePlag eine Software entwickelt, die sowohl die Kandidatensuche anhand einer lokalen Dokumentenbasis, als auch das Text Alignment beinhaltet. Es wurde zu Beginn der Planung der Software diskutiert, ob eine Internetsuche nach Kandidatendokumenten implementiert werden solle. Diese Frage wurde aufgeschoben und wir einigten uns schließlich zunächst bei der rein lokalen Dokumentenbasis zu bleiben.

Das Ziel eine Anwendung für die Plagiatsfindung zu entwickeln wurde erreicht. SciencePlag erzielte gute Resultate bei der Plagiatssuche im PAN Trainingskorpus. Dies legt die Vermutung nahe, dass sich SciencePlag auch gut im direkten Vergleich zu den Teilnehmern des PAN Wettbewerbs schlägt. Der direkte Vergleich konnte leider nicht durchgeführt werden, da die benötigten Testsysteme und Testdaten nicht zur Verfügung standen.

Es wurde im Laufe der PG die Verarbeitung von PDF-Dokumenten zu implementieren angedacht, jedoch erwies sich schon bei den ersten Betrachtungen diese Aufgabe als zu komplex und zeitaufwändig für den Rahmen dieser PG. Daher beschränkten wir uns auf die vom PAN-Wettbewerb übernommene Anforderung der Verarbeitung von reinen Text Dokumenten. Dies reichte für unsere Anwendungsfälle, insbesondere die Verarbeitung der Trainingsdaten des PAN-Wettbewerbs aus, jedoch sollte eine alltagstaugliche Plagiatssoftware auch PDF-Dokumente verarbeiten können.

SciencePlag wurde bewusst als reine Konsolenanwendung konzipiert, dies ermöglicht eine einfache Nutzung durch den Anwender. Nachteil dabei ist, dass sich die Ausgabe der Ergebnisse in der Konsole nicht besonders übersichtlich gestaltet. Aus diesem Grund haben wir die Javascript basierte Webanwendung für die Darstellung der Ergebnisse entwickelt. Diese stellt die Resultate für den Anwender übersichtlich und nachvollziehbar dar.

Insgesamt betrachtet, wurden im Rahmen der Projektgruppe SciencePlag vielversprechende und zukunftsorientierte Ideen und Vorgehensweisen für die Plagiatssuche mit Hilfe von Information Retrieval und Suffixarrays entwickelt und in Form der Software SciencePlag praktisch umgesetzt.

7.2 Ablauf der PG

Zu Beginn der Projektgruppe wurden die Grundlagen der für die Entwicklung relevanten Themenbereiche in einer Seminarphase aufgearbeitet. Dabei stellten die Mitglieder jeweils einen Themenbereich ausführlich vor. Die Themen umfassten einen Überblick über die Indexstrukturen und Algorithmen der Stringology, das Information Retrieval und zugehörige Bewertungsmaße. Weiterhin wurden Textvorverabeitung, wie Stammformreduktion und Stoppwortentfernung, sowie parallele und externe Algorithmen behandelt. Neben diesen rein technischen Bereichen wurden verschiedene Techniken des Projektmanagements vorgestellt. Wir einigten uns auf ein iteratives Vorgehensmodell mit Anlehnungen an *Scrum*. Es wurde ein vom ITMC zur Verfügung gestelltes Redmine-System eingerichtet, welches uns u.a. Ticketsystem, Wiki und Versionskontrolle bereit stellte. Die folgende Arbeit der Projektgruppe lässt sich in fünf Phasen einteilen, welche in etwa den von uns festgelegten Milestones entsprechen:

- *Entwicklung des Algorithmus*: Es wurde die grundlegende Ideen der geplanten Software entworfen. Hier wurde der Grundstein für den zweiteiligen Ablauf der Plagiatssuche durch Candidate Retrieval und Text Alignment gelegt. Nachdem diese Teilung feststand, wurde eine Aufteilung der PG-Teilnehmer auf die Bereiche vorgenommen, so dass die beiden Gruppen parallel die Details der Schritte ausarbeiten konnten. Die Ergebnisse wurden dabei ausführlich dokumentiert.

- *Planung der Softwarearchitektur*: Wir entschieden uns schnell für einen möglichst modularen Aufbau der Software, was auch die Arbeit in den festgelegten Gruppen erleichterte. Dazu wurden in einer Reihe von Treffen die benötigten Schnittstellen zwischen Candidate Retrieval und Text Alignment bestimmt.

- *Fertigstellung einer ersten Version*: Nachdem die Planung der Architektur abgeschlossen war, entschieden wir uns einen ersten Prototyp bis zum Ende des ersten PG-Semesters fertigzustellen. Dieses Ziel wurde erreicht und es wurde begonnen Ideen zur Verbesserungen zu sammeln.

- *Fertigstellung der finalen Version*: Zu Beginn des zweites Semesters der PG wurden die wichtigsten offenen Punkte und zu behebenden Fehler gesammelt. Es wurden

zunächst die bereits bekannten Fehler behoben und anschließend weitere Verbesserungen an den Algorithmen, der Usability und der Ausgabe der Resultate vorgenommen.

- *Evaluation*: Ein wichtiger Punkt in jeder wissenschaftlichen Arbeit ist die Evaluation. Deswegen wurden ab Mitte des zweiten Semesters ein Großteil der Ressourcen der PG auf diese gelenkt. Evaluiert wurde u.a. anhand der Trainingsdaten des PAN-Wettbewerbs 2013. Für die automatische Auswertung der Ergebnisse wurden Python Skripte geschrieben, welche die Evaluationsschritte in Teilen automatisierten.

Es gab teilweise Überlagerungen dieser Phasen sowie weitere Überarbeitungen bereits erreichter Teilziele. Dies war durch das gewählte iterative Vorgehensmodell aber durchaus gewünscht.

7.3 Fazit und Ausblick

Die Möglichkeiten Plagiate zu verhindern sind begrenzt, Plagiarismus wird es vielleicht immer geben und damit wohl auch immer den Bedarf an guter Software zur Entdeckung dieser Plagiate. Die Projektgruppe SciencePlag hat einen neuen Weg der Plagiatssuche mit Hilfe von Algorithmen der Stringology und des Information Retrieval beschritten. Es wurden dabei vielversprechende Ergebnisse erreicht. Für einen weitreichenden Einsatz unter realen Bedingungen, fern vom *Experimentiertisch* der PG, bedarf es jedoch etwas weitere Arbeit und Änderungen an SciencePlag. Insbesondere muss die bereits erwähnte Unterstützung weiterer Dateiformate, wie z. B. PDF, implementiert werden.

Gerade in Zeiten der fortschreitenden Digitalisierung stehen beiden Seiten, den Plagiateuren sowie denen die Plagiarismus bekämpfen, mehr und mehr Daten zur Verfügung. Einerseits wird es einfacher, Texte zu kopieren und diese sich zu eigen zu machen, andererseits ermöglicht dies auch die automatisierte Suche nach Textfragmenten im Internet und erleichtert somit das Entdecken von Plagiaten. Diese grundlegende Funktionalität, die Suche nach neuen Kandidaten-Dokumenten im Internet, würde den Nutzen und die Praktikabilität von SciencePlag enorm erhöhen und ggf. zu noch umfangreicheren Ergebnissen führen.

Auf algorithmischer Ebene sollte eine Verfeinerung des Candidate Retrieval vorgenommen werden, so dass bei diesem Schritt nicht mehr ganze Dokumente sondern Textabschnitte betrachtet werden. Eine natürliche Aufteilung in vorhandene Kapitel und Abschnitte wäre hier denkbar. Diese feinere Suche sollte zu einer besseren Auswahl an Kandidaten führen und somit möglicherweise zu einem insgesamt besseren Resultat. Eine weitere Idee ist die Erweiterung der Integer-Repräsentation durch Synonyme, um Verschleierungen durch solche besser zu erkennen. Dazu wird die Abbildung eines einzelnen Wortes auf einen Integer durch die Abbildung einer Menge von Wörtern auf diesen Integer ersetzt. In dieser Menge befinden sich dabei die Synonyme, welche ein identisches oder ähnliches semantisches Konzept beschreiben. So kann z. B. das Wort *Auto* durch die Menge *{Auto, Automobil, Fahrzeug}* erweitert werden.

Um viele mögliche Nutzer zu erreichen, kann eine *cloudbasierte* Version von SciencePlag entwickelt werden, die es ermöglicht Plagiatssuche über eine Website durchzuführen. Da die Ausgabe der Ergebnisse bereits durch die Javascript Viewer Anwendung realisiert wurde, müsste diese noch um eine geeignetes Anfrage-Formular erweitert werden. Dafür wäre eine Anpassung von SciencePlag zur Verwendung mit einem geeigneten Backend notwendig. Ein solches SciencePlag-Backend könnte zudem Webservices für Plagiatssuche zur Verfügung stellen, was die Integration von SciencePlag in weitere Anwendungen, in denen Plagiarismus eine Rolle spielt, ermöglichen würde.

ANHANG A

Entwicklungsumgebung

Im folgenden Abschnitt werden die wichtigsten Bibliotheken und Tools vorgestellt, die für die Entwicklung von SciencePlag verwendet wurden. Dabei wird kurz die Funktion erläutert sowie eine Begründung für die Verwendung gegeben. Für weitere Informationen zu den Bibliotheken sei der interessierte Leser auf die jeweils verlinkten Homepages verwiesen. Zuerst wird noch auf die gewählte Programmiersprache C++ und auf die Versionsverwaltung mit Git eingegangen.

Git

Zur Versionsverwaltung wird *Git* verwendet. Git zeichnet sich gegenüber den älteren System wie z.B. Subversion(SVN) durch eine größere Flexibilät aus bei der Arbeit in größeren Teams. Hervorzuheben ist dabei die Möglichkeit lokal zu arbeiten, und Änderungen nur freigezugeben, wenn diese als fertig anzusehen sind. Um den Quellcode aus dem Git Repository zu laden, muss Git zuerst installiert werden. Eine Installationsanleitung für die unterschiedlichen Betriebssystem ist auf der Homepage von Git[1] zu finden. Anschließend kann der Quellcode mit dem folgenden Befehl abgerufen werden.

```
git clone ssh://git@projekte.itmc.tu-dortmund.de/in/pg588.git
```

Innerhalb des Repository ist der Ordner *SciencePlag* das Projektverzeichnis.

Programmiersprache

Für die Entwicklung von SciencePlag haben wir uns auf die Programmiersprache C++ geeinigt. Grund dafür ist der Vorteil von höherer Performanz, den C++ gegenüber anderen Hochsprachen wie Python hat. Ein Test von Google [9] hat dies z.B. neulich erneut gezeigt. Insbesondere ist ein schnelles Arbeiten auf der im Abschnitt 4.2 beschriebene Datenbasis erforderlich, da diese potenziell sehr groß werden kann.

Des Weiteren bietet C++ im Vergleich zu C mehr Unterstützung für moderne Entwicklungskonzepte wie objektorientierte Programmierung an.

[1] `https://git-scm.com/` (Zuletzt abgerufen: 06.04.2016)

Codingstyle-Conventions

Um eine einfache Einarbeitung in den Code, z.B. bei einer Weiterentwicklung nach dem Ende der Projektgruppe, zu ermöglichen, wurde ein einfacher, weit verbreiteter Codingstyle gesucht. Dabei ist die Entscheidung auf den von Google beschriebenen Codingstyle[2] gefallen. Ein zusätzlicher Vorteil ist das publizierte Pythonscript, dass den geschriebenen Quellcode auf die Einhaltung der meisten Punkte kontrolliert.

Buildtool CMake

CMake wurde von uns als Buildtool ausgewählt. Grundlage für die Entscheidung war die Tatsache das einige Projektgruppen Teilnehmer bereits mit *CMake*[3] vertraut waren.

CMake erzeugt gesteuert durch Skript Dateien, genannt *CMakeLists.txt*, Makefiles. Dabei prüft CMake automatisch Abhängigkeiten innerhalb des Projekts und zu den benötigten Bibliotheken. Diese können dann automatisch geladen und kompiliert werden. Zuletzt werden die Header Dateien in das Makefile aufgenommen. Alle diese Schritte werden ohne weitere Eingaben des Anwenders ausgeführt.

Doxygen

Für die Dokumentation des Quelltextes haben wir uns auf *Doxygen*[4] geeinigt. Doxygen erzeugt aus speziell gekennzeichnenten Kommentaren in den Quelldateien Dokumentation in verschiedenen Ausgabeformaten, sowie einen Überblick über die Vererbungshierarchie.

Doxygen kann mit verschiedenen Kommentarsyntaxen umgehen. Die Wahl innerhalb der Projektgruppe ist auf den Qt-Style gefallen. Quelltext A.1 zeigt ein Beispiel für diesen.

```
1 //! Aligns the suspicious and all candidate documents.
2 /*!
3   Aligns the suspicious document and every candidate document.
4   All suspicious passages identified as plagiarism will be returned.
5   \param the suspicious document
6   \param the cadidate documents as vector
7   \return a vector of results representing all identified plagiarisms.
8 */
```

Quelltext A.1: Beispiel des Qt Styles für die Doxygen Dokumentation.

Logging

Um unterschiedliche Loggingausgaben und Granularität zu erlauben wurde eine Loggingbibliothek benötigt. Dabei ist die Entscheidung auf *Easylogging++*[5] gefallen. Ausschlaggebender Grund war die einfache Einbindung, die sich daraus ergibt, dass Easylogging++ nur aus einer C++-Header-Datei besteht.

Um eine feinere Granularität beim Logging erhalten zu können, wurden die von Easylogging++ bereitgestellten Makros um vier unterschiedliche Suffixe erweitert, die dafür sorgen, dass für jede dieser Suffixe unterschiedliche Logginglevel gesetzt werden können:

[2] `https://google-styleguide.googlecode.com/svn/trunk/cppguide.html` (Zuletzt abgerufen: 06.04.2016)

[3] `http://www.cmake.org` (Zuletzt abgerufen: 06.04.2016)

[4] `http://www.stack.nl/~dimitri/doxygen/`

[5] `http://easylogging.muflihun.com/` (Zuletzt abgerufen: 06.04.2016)

```
_PRE     = Preprocessing
_CR      = Candidate Retrieval
_ALIGN   = Text Alignment
_SPLAG   = Splag-Konsolenprogramm
```

Auf diese Weise können bei der Entwicklung störende Logging-Ausgaben anderer Komponenten der Software abgeschaltet werden. Die Konfiguration der Logginglevels ist in Abschnitt 5.3.5 beschrieben.

Unit-Tests

Für die Implementierung und Ausführung von Unit-Tests haben wir uns für das *Google Test*[6] Framework von Google entschieden. Ein Grund dafür war zum einen die komfortable Bedienung, wie z.B. die automatische Test-Registrierung. Des Weiteren hatten mehrere Mitglieder aus unserer Projektgruppe bereits Erfahrungen mit Google Test sammeln können.

Für die Entwicklung von SciencePlag haben wir uns darauf geeinigt, dass Unit-Tests für alle Schnittstellen zu implementieren sind. Innerhalb der Komponenten, ist dies dem jeweiligen Entwickler bzw. der Kleingruppe überlassen. Im Allgemeinen wurde aber für die Teilkomponenten und Algorithmen jeweils ein Unit-Test implementiert.

Programmoptionen

Für eine einfache Verwaltung der verschiedenen Optionsmöglichkeiten für SciencePlag haben wir uns für die Bibliothek *program_options* aus der weitverbreiteten Bibliothekssammlung *Boost* [22] entschieden. In Boost finden sich viele verschiedene Bibliotheken, die sich das Ziel setzen, einen Quasi-Standard für nicht in der C++-Standardbibliothek enthaltene Funktionalitäten zu bieten und möglichst reibungslos mit der vorhandenen Standardbibliothek zu arbeiten.

program_options hat sich hier nicht nur durch die einfache Einbindung und die übersichtliche Schnittstelle bewährt, sondern auch durch die Fähigkeiten, Konfigurationsdateien in einem sehr ähnlichen Schema einzulesen (siehe Abschnitt 5.3.3). Fehlerhafte Nutzung durch fehlende oder falsche Programmoptionen wird direkt und klar an den Nutzer weitergegeben werden (siehe Abschnitt 5.3.1), sodass dieser intuitiv mit SciencePlag arbeiten kann.

[6] `https://code.google.com/p/googletest/` (Zuletzt abgerufen: 06.04.2016)

ANHANG B

Installation

In diesem Abschnitt werden die einzelnen Schritte aufgeführt, damit SciencePlag kompiliert werden kann:

Abhängigkeiten

Zu Beginn werden die Linux-Pakete installiert, die von SciencePlag verwendet werden. Dazu wird jeweils kurz eine Erläuterung gegeben, wofür diese benötigt werden.

- `subversion`
  Google Test (Abschnitt A) wird beim kompilieren direkt vom Google-Code-Subversion-Server geladen.
- `libboost-all-dev/boost/devel` *Version:* $\geq$ 1.37.0
  Aus der Boost-Bibliothek werden die `programoptions` (Abschnitt A), sowie die `filesystem`-Komponente benötigt.
- `python2`
  Die Verwendung von Python ist optional, da nur das Wikipedia-Script in Python geschrieben ist.
- `libsqlite3-dev` Die Datenbasis wird mithilfe von sqlite gespeichert und verwaltet.
- `cmake:` *Version:* $\geq$ 3.2.2
  CMake (Abschnitt: A) wird zum Kompilieren von SciencePlag verwendet.

SciencePlag kompilieren

Um SciencePlag zu kompilieren, wird wie folgt vorgegangen:

- In der Konsole ins Projektverzeichnis wechseln
- In den Ordner *build* wechseln. Wenn nicht vorhanden, erstellen
- Makefile mit cmake generieren: *cmake ..*
- SciencePlag kompilieren: *make*
- Die Verwendung ist in Kapitel 5 beschrieben

Der erste Kompilierungsvorgang kann einige Minuten in Anspruch nehmen, da Google Test heruntergeladen und kompiliert werden muss.

Output Viewer einrichten

Im Folgenden wird ausgeführt, welche Schritte nötig sind, um den Output Viewer lokal ausführen zu können.

Abhängigkeiten

Um Output Viewer lokal zu starten, müssen die folgenden Abhängigkeiten installiert werden:

- *NodeJS* (`https://nodejs.org/en/`)
  NodeJS wird nicht benötigt, lediglich dessen Package-Manager *npm*.
- Ruby und Sass
  Sass ist in Ruby geschrieben, daher wird auch Ruby benötigt. Sass ist ein CSS-Pre-Compiler, der auch Vererbung, Variablen usw. unterstützt. Eine Installationsanleitung ist unter `http://sass-lang.com/` zu finden.
- Bower
  Verwaltet die Skript Bibliotheken. Installation: *npm install bower -g*
- Grunt
  Grunt ist ein Buildsystem für Web-Projekte. Damit werden einzelne Schritte (wie z.B. sass) automatisiert. Installation: *npm install grunt -g*

JavaScript Bibliotheken

Nach der Installation der obigen Abhängigkeiten, können die Bibliotheken installiert werden, die für Output Viewer benötigt werden. Dafür wird mit der Konsole ins Projektverzeichnis gewechselt und die folgenden Befehle ausgeführt:
npm install
bower install
Bower ist ein Paket-Verwaltungsystem, das mit dem *install* Befehl automatisch alle in der *bower.json* Datei aufgeführten Bibliotheken installiert. Für Output Viewer sind es diese:

- angular `https://angularjs.org/`
- angular-chart.js
  `http://jtblin.github.io/angular-chart.js/`
- angular-cookies
  `https://angularjs.org/`
- angular-material
  `https://material.angularjs.org/latest/`
- angular-sanitize
  `https://angularjs.org/`
- angular-slimscroll
  `https://github.com/ziscloud/angular-slimscroll`
- angular-translate
  `https://github.com/angular-translate/angular-translate`
- angular-translate-loader-partial
  `https://github.com/angular-translate/angular-translate`
- angular-translate-storage-cookie
  `https://github.com/angular-translate/angular-translate`

- angular-translate-storage-local
  `https://github.com/angular-translate/angular-translate`
- angular-ui-router
  `https://github.com/angular-ui/ui-router`
- angular-xml
  `https://github.com/johngeorgewright/angular-xml`
- chartjs
  `http://www.chartjs.org/`
- jquery
  `https://jquery.com/`
- material-design-iconic-font
  `http://zavoloklom.github.io/material-design-iconic-font/`
- nsPopover
  `https://github.com/nohros/nsPopover`

Webserver starten

grunt serve Damit wird ein Webserver gestartet, der unter `http://localhost:9000/#/compare/splag` erreichbar ist.

ANHANG C

Stoppwörter

Im Folgenden sind die von uns im Vorverarbeitungsschritt verwendeten Stoppwörter nach Sprache getrennt gelistet.

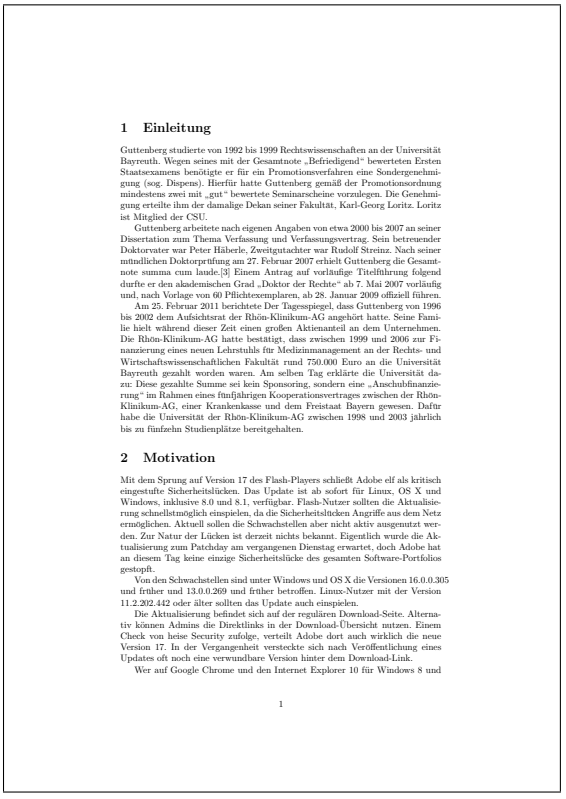
Deutsche Stoppwörter: aber, alle, allem, allen, aller, alles, als, also, am, an, ander, andere, anderem, anderen, anderer, anderes, anderm, andern, anderr, anders, auch, auf, aus, bei, bin, bis, bist, da, damit, dann, der, den, des, dem, die, das, daß, derselbe, derselben, denselben, desselben, demselben, dieselbe, dieselben, dasselbe, dazu, dein, deine, deinem, deinen, deiner, deines, denn, derer, dessen, dich, dir, du, dies, diese, diesem, diesen, dieser, dieses, doch, dort, durch, ein, eine, einem, einen, einer, eines, einig, einige, einigem, einigen, einiger, einiges, einmal, er, ihn, ihm, es, etwas, euer, eure, eurem, euren, eurer, eures, für, gegen, gewesen, hab, habe, haben, hat, hatte, hatten, hier, hin, hinter, ich, mich, mir, ihr, ihre, ihrem, ihren, ihrer, ihres, euch, im, in, indem, ins, ist, jede, jedem, jeden, jeder, jedes, jene, jenem, jenen, jener, jenes, jetzt, kann, kein, keine, keinem, keinen, keiner, keines, können, könnte, machen, man, manche, manchem, manchen, mancher, manches, mein, meine, meinem, meinen, meiner, meines, mit, muss, musste, nach, nicht, nichts, noch, nun, nur, ob, oder, ohne, sehr, sein, seine, seinem, seinen, seiner, seines, selbst, sich, sie, ihnen, sind, so, solche, solchem, solchen, solcher, solches, soll, sollte, sondern, sonst, über, um, und, uns, unse, unsem, unsen, unser, unses, unter, viel, vom, von, vor, während, war, waren, warst, was, weg, weil, weiter, welche, welchem, welchen, welcher, welches, wenn, werde, werden, wie, wieder, will, wir, wird, wirst, wo, wollen, wollte, würde, würden, zu, zum, zur, zwar, zwischen

Englische Stoppwörter: i, me, my, myself, we, our, ours, ourselves, you, your, yours, yourself, yourselves, he, him, his, himself, she, her, hers, herself, it, its, itself, they, them, their, theirs, themselves, what, which, who, whom, this, that, these, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing, a, an, the, and, but, if, or, because, as, until, while, of, at, by, for, with, about, against, between, into, through, during, before, after, above, below, to, from, up, down, in, out, on, off, over, under, again, further, then, once, here, there, when, where, why, how, all, any, both, each, few, more, most, other, some, such, no, nor, not, only, own, same, so, than, too, very, s, t, can, will, just, don, should, now

ANHANG D

Testdokumente für Plagiatssoftware

Selbst angefertigtes Plagiat



1 Einleitung

Guttenberg studierte von 1992 bis 1999 Rechtswissenschaften an der Universität Bayreuth. Wegen seines mit der Gesamtnote „Befriedigend“ bewerteten Ersten Staatsexamens benötigte er für ein Promotionsverfahren eine Sondergenehmigung (sog. Dispens). Hierfür hatte Guttenberg gemäß der Promotionsordnung mindestens zwei mit „gut“ bewertete Seminarscheine vorzulegen. Die Genehmigung erteilte ihm der damalige Dekan seiner Fakultät, Karl-Georg Loritz. Loritz ist Mitglied der CSU.

Guttenberg arbeitete nach eigenen Angaben von etwa 2000 bis 2007 an seiner Dissertation zum Thema Verfassung und Verfassungsvertrag. Sein betreuender Doktorvater war Peter Häberle, Zweitgutachter war Rudolf Streinz. Nach seiner mündlichen Doktorprüfung am 27. Februar 2007 erhielt Guttenberg die Gesamtnote summa cum laude.[3] Einem Antrag auf vorläufige Titelführung folgend durfte er den akademischen Grad „Doktor der Rechte“ ab 7. Mai 2007 vorläufig und, nach Vorlage von 60 Pflichtexemplaren, ab 28. Januar 2009 offiziell führen.

Am 25. Februar 2011 berichtete Der Tagesspiegel, dass Guttenberg von 1996 bis 2002 dem Aufsichtsrat der Rhön-Klinikum-AG angehört hatte. Seine Familie hielt während dieser Zeit einen großen Aktienanteil an dem Unternehmen. Die Rhön-Klinikum-AG hatte bestätigt, dass zwischen 1999 und 2006 zur Finanzierung eines neuen Lehrstuhls für Medizinmanagement an der Rechts- und Wirtschaftswissenschaftlichen Fakultät rund 750.000 Euro an die Universität Bayreuth gezahlt worden waren. Am selben Tag erklärte die Universität dazu: Diese gezahlte Summe sei kein Sponsoring, sondern eine „Anschubfinanzierung“ im Rahmen eines fünfjährigen Kooperationsvertrages zwischen der Rhön-Klinikum-AG, einer Krankenkasse und dem Freistaat Bayern gewesen. Dafür habe die Universität der Rhön-Klinikum-AG zwischen 1998 und 2003 jährlich bis zu fünfzehn Studienplätze bereitgehalten.

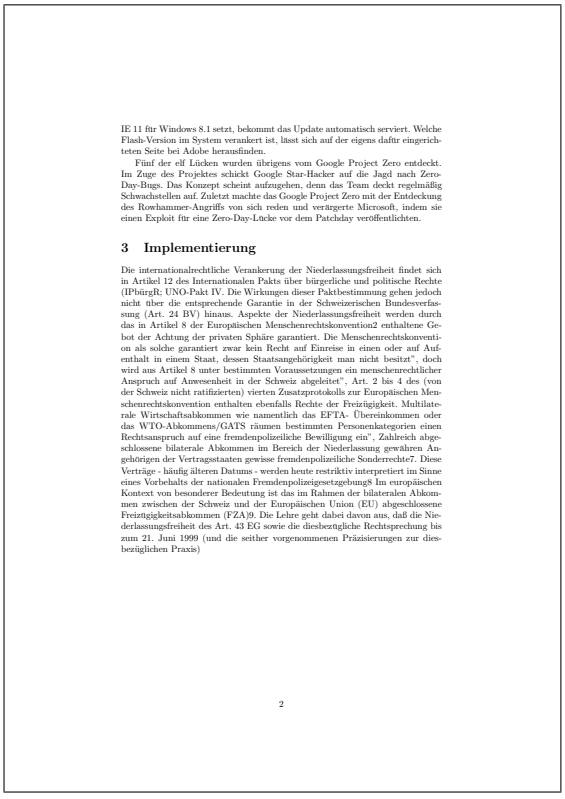
2 Motivation

Mit dem Sprung auf Version 17 des Flash-Players schließt Adobe elf als kritisch eingestufte Sicherheitslücken. Das Update ist ab sofort für Linux, OS X und Windows, inklusive 8.0 und 8.1, verfügbar. Flash-Nutzer sollten die Aktualisierung schnellstmöglich einspielen, da die Sicherheitslücken Angriffe aus dem Netz ermöglichen. Aktuell sollen die Schwachstellen aber nicht aktiv ausgenutzt werden. Zur Natur der Lücken ist derzeit nichts bekannt. Eigentlich wurde die Aktualisierung zum Patchday am vergangenen Dienstag erwartet, doch Adobe hat an diesem Tag keine einzige Sicherheitslücke des gesamten Software-Portfolios gestopft.

Von den Schwachstellen sind unter Windows und OS X die Versionen 16.0.0.305 und früher und 13.0.0.269 und früher betroffen. Linux-Nutzer mit der Version 11.2.202.442 oder älter sollten das Update auch einspielen.

Die Aktualisierung befindet sich auf der regulären Download-Seite. Alternativ können Admins die Direktlinks in der Download-Übersicht nutzen. Einem Check von heise Security zufolge, verteilt Adobe dort auch wirklich die neue Version 17. In der Vergangenheit versteckte sich nach Veröffentlichung eines Updates oft noch eine verwundbare Version hinter dem Download-Link.

Wer auf Google Chrome und den Internet Explorer 10 für Windows 8 und



IE 11 für Windows 8.1 setzt, bekommt das Update automatisch serviert. Welche Flash-Version im System verankert ist, lässt sich auf der eigens dafür eingerichteten Seite bei Adobe herausfinden.

Fünf der elf Lücken wurden übrigens vom Google Project Zero entdeckt. Im Zuge des Projektes schickt Google Star-Hacker auf die Jagd nach Zero-Day-Bugs. Das Konzept scheint aufzugehen, denn das Team deckt regelmäßig Schwachstellen auf. Zuletzt machte das Google Project Zero mit der Entdeckung des Rowhammer-Angriffs von sich reden und verärgerte Microsoft, indem sie einen Exploit für eine Zero-Day-Lücke vor dem Patchday veröffentlichten.

3 Implementierung

Die internationalrechtliche Verankerung der Niederlassungsfreiheit findet sich in Artikel 12 des Internationalen Pakts über bürgerliche und politische Rechte (IPbürgR; UNO-Pakt IV. Die Wirkungen dieser Paktbestimmung gehen jedoch nicht über die entsprechende Garantie in der Schweizerischen Bundesverfassung (Art. 24 BV) hinaus. Aspekte der Niederlassungsfreiheit werden durch das in Artikel 8 der Europäischen Menschenrechtskonvention2 enthaltene Gebot der Achtung der privaten Sphäre garantiert. Die Menschenrechtskonvention als solche garantiert zwar kein Recht auf Einreise in einen oder auf Aufenthalt in einem Staat, dessen Staatsangehörigkeit man nicht besitzt“, doch wird aus Artikel 8 unter bestimmten Voraussetzungen ein menschenrechtlicher Anspruch auf Anwesenheit in der Schweiz abgeleitet“, Art. 2 bis 4 des (von der Schweiz nicht ratifizierten) vierten Zusatzprotokolls zur Europäischen Menschenrechtskonvention enthalten ebenfalls Rechte der Freizügigkeit. Multilaterale Wirtschaftsabkommen wie namentlich das EFTA- Übereinkommen oder das WTO-Abkommens/GATS räumen bestimmten Personenkategorien einen Rechtsanspruch auf eine fremdenpolizeiliche Bewilligung ein“, Zahlreich abgeschlossene bilaterale Abkommen im Bereich der Niederlassung gewähren Angehörigen der Vertragstaaten gewisse fremdenpolizeiliche Sonderrechte7. Diese Verträge - häufig älteren Datums - werden heute restriktiv interpretiert im Sinne eines Vorbehalts der nationalen Fremdenpolizeigesetzgebung8 Im europäischen Kontext von besonderer Bedeutung ist das im Rahmen der bilateralen Abkommen zwischen der Schweiz und der Europäischen Union (EU) abgeschlossene Freizügigkeitsabkommen (FZA)9. Die Lehre geht dabei davon aus, daß die Niederlassungsfreiheit des Art. 43 EG sowie die diesbezügliche Rechtsprechung bis zum 21. Juni 1999 (und die seither vorgenommenen Präzisierungen zur diesbezüglichen Praxis)

Generiertes Paper

Download a Postscript or PDF version of this paper.
Download all the files for this paper as a gzipped tar archive.
Generate another one.
Back to the SCIgen homepage.

Deconstructing the Location-Identity Split

Ole Bergenholtz

Abstract

The implications of classical technology have been far-reaching and pervasive. Although it at first glance seems perverse, it has ample historical precedence. In this paper, we show the investigation of link-level acknowledgements. *Mete*, our new system for the exploration of online algorithms, is the solution to all of these challenges [12,18].

Table of Contents

1 Introduction

In recent years, much research has been devoted to the investigation of Smalltalk; nevertheless, few have visualized the exploration of SCSI disks. Without a doubt, it should be noted that *Mete* manages compilers. The shortcoming of this type of solution, however, is that operating systems can be made unstable, "fuzzy", and encrypted. Contrarily, gigabit switches alone can fulfill the need for RPCs.

We use modular communication to argue that kernels [6] and multicast methodologies are generally incompatible. We view complexity theory as following a cycle of four phases: visualization, allowance, visualization, and location. The shortcoming of this type of method, however, is that the little-known extensible algorithm for the investigation of the World Wide Web by Ito et al. runs in $\Omega(n)$ time. The basic tenet of this approach is the refinement of systems. Contrarily, this approach is mostly well-received. Existing flexible and relational heuristics use the evaluation of Web services to store the evaluation of active networks.

The rest of this paper is organized as follows. We motivate the need for IPv6. Further, we verify the simulation of systems. In the end, we conclude.

2 Related Work

In designing our application, we drew on related work from a number of distinct areas. Similarly, despite the fact that Richard Hamming also presented this method, we synthesized it independently and simultaneously [41,41]. It remains to be seen how valuable this research is to the software engineering community. Wilson [24,18,36] developed a similar framework, nevertheless we proved that our heuristic follows a Zipf-like distribution. Even though this work was published before ours, we came up with the approach first but could not publish it until now due to red tape. Unlike many related methods [8,15,28], we do not attempt to study or prevent reliable symmetries. This work follows a long line of related applications, all of which have failed. Thompson proposed several

extensible methods [9], and reported that they have profound impact on self-learning technology. Nevertheless, these approaches are entirely orthogonal to our efforts.

Though Bose et al. also introduced this solution, we improved it independently and simultaneously. Continuing with this rationale, a recent unpublished undergraduate dissertation [32] introduced a similar idea for semaphores. Watanabe et al. [23,11,25] and H. Bose motivated the first known instance of consistent hashing [9]. Ultimately, the algorithm of Anderson et al. [35,36,30,10] is a significant choice for real-time information [38].

Our solution is related to research into active networks, the understanding of scatter/gather I/O, and hierarchical databases [9,17,7]. A litany of previous work supports our use of ubiquitous symmetries [20,8,42,16,5,38,40]. Next, unlike many previous methods [29], we do not attempt to manage or study the refinement of RPCs [4,39,31,26,33]. This work follows a long line of existing systems, all of which have failed. Furthermore, a litany of prior work supports our use of authenticated information [13]. It remains to be seen how valuable this research is to the programming languages community. These methods typically require that virtual machines and interrupts are regularly incompatible [1], and we proved in our research that this, indeed, is the case.

3 Framework

Mete relies on the structured model outlined in the recent little-known work by Christos Papadimitriou in the field of cyberinformatics. Rather than caching the visualization of operating systems, *Mete* chooses to locate omniscient technology. This is a theoretical property of our solution. Therefore, the architecture that *Mete* uses is solidly grounded in reality [1].

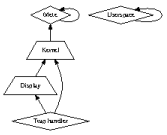


Figure 1: The schematic used by our methodology.

Suppose that there exists virtual machines such that we can easily deploy red-black trees. The framework for our methodology consists of four independent components: DHTs, replication, reinforcement learning, and constant-time symmetries. This is a practical property of *Mete*. We believe that web browsers and SCSI disks can cooperate to realize this ambition. Despite the fact that experts largely estimate the exact opposite, our framework depends on this property for correct behavior. Thus, the framework that *Mete* uses is not feasible.

4 Implementation

Mete is elegant; so, too, must be our implementation. We have not yet implemented the server

daemon, as this is the least technical component of *Mete*. On a similar note, *Mete* is composed of a centralized logging facility, a client-side library, and a server daemon. Even though we have not yet optimized for complexity, this should be simple once we finish coding the server daemon. We have not yet implemented the server daemon, as this is the least confirmed component of our framework [34]. We plan to release all of this code under Microsoft's Shared Source License.

5 Results

Building a system as unstable as our would be for naught without a generous performance analysis. We did not take any shortcuts here. Our overall performance analysis seeks to prove three hypotheses: (1) that RPCs no longer impact distance; (2) that checksums no longer affect system design; and finally (3) that randomized algorithms have actually shown weakened 10th-percentile time since 2004 over time. We are grateful for separated write-back caches; without them, we could not optimize for usability simultaneously with usability. Our evaluation strives to make these points clear.

5.1 Hardware and Software Configuration

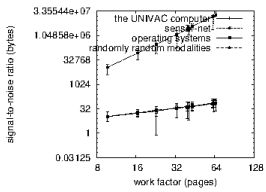


Figure 2: The effective complexity of our heuristic, as a function of bandwidth.

One must understand our network configuration to grasp the genesis of our results. We performed a packet-level deployment on our classical cluster to disprove the opportunistically secure nature of adaptive models. This outcome at first glance seems unexpected but is derived from known results. To begin with, we removed 3Gb/s of Ethernet access from our embedded cluster to examine the tape drive speed of our mobile telephones. Configurations without this modification showed duplicated throughput. We reduced the effective flash-memory speed of the KGB's mobile telephones to probe configurations. We added more 7GHz Pentium IIIs to the NSA's underwater testbed to understand the effective USB key throughput of our underwater testbed [37,22,3,2]. On a similar note, we added some 100GHz Pentium IVs to the KGB's desktop machines to measure J. Ullman's deployment of context-free grammar in 1986. This step flies in the face of conventional wisdom, but is instrumental to our results. Similarly, Russian computational biologists reduced the tape drive space of our desktop machines to discover our network [21]. Lastly, we doubled the seek time of our collaborative testbed to quantify the mutually metamorphic behavior of independent modalities.

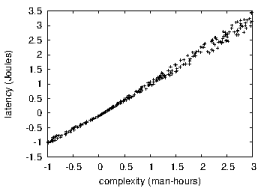


Figure 3: The average interrupt rate of *Mete*, compared with the other heuristics [27].

Mete does not run on a commodity operating system but instead requires a lazily hardened version of LeOS. All software components were compiled using AT&T System V's compiler built on Leslie Lamport's toolkit for computationally exploring saturated wide-area networks. We implemented our courseware server in ML, augmented with topologically separated extensions. Furthermore, our experiments soon proved that automating our mutually exclusive Atari 2600s was more effective than patching them, as previous work suggested. All of these techniques are of interesting historical significance; R. Tarjan and K. Martinez investigated a similar setup in 2001.

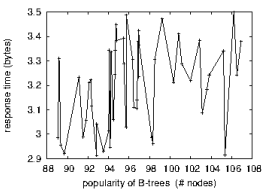


Figure 4: The effective block size of *Mete*, as a function of response time.

5.2 Experimental Results

Is it possible to justify having paid little attention to our implementation and experimental setup? Yes. Seizing upon this contrived configuration, we ran four novel experiments: (1) we deployed 16 Atari 2600s across the planetary-scale network, and tested our multi-processors accordingly; (2) we compared popularity of Markov models on the Sprite, OpenBSD and AT&T System V operating systems; (3) we compared seek time on the Microsoft Windows 1969, TinyOS and Ultrix operating systems; and (4) we dogfooded our heuristic on our own desktop machines, paying particular attention

to flash-memory throughput [14].

We first analyze the second half of our experiments as shown in Figure 4. Operator error alone cannot account for these results. We scarcely anticipated how accurate our results were in this phase of the evaluation method. The curve in Figure 4 should look familiar; it is better known as $H_Y(n) = n$.

We next turn to experiments (1) and (4) enumerated above, shown in Figure 2. Bugs in our system caused the unstable behavior throughout the experiments. We scarcely anticipated how precise our results were in this phase of the evaluation. These median response time observations contrast to those seen in earlier work [15], such as Manuel Blum's seminal treatise on operating systems and observed instruction rate.

Lastly, we discuss experiments (1) and (4) enumerated above. Note that Figure 2 shows the *median* and not *10th-percentile* partitioned bandwidth. Similarly, the many discontinuities in the graphs point to degraded mean clock speed introduced with our hardware upgrades. Third, error bars have been elided, since most of our data points fell outside of 59 standard deviations from observed means.

6 Conclusion

To address this problem for randomized algorithms, we explored new multimodal symmetries. To surmount this obstacle for hash tables, we proposed a probabilistic tool for architecting IPv6 [19]. Next, we demonstrated that simplicity in our algorithm is not an obstacle. In fact, the main contribution of our work is that we examined how SCSI disks can be applied to the investigation of web browsers. This is crucial to the success of our work. The study of sensor networks is more key than ever, and *Mete* helps end-users do just that.

References

[1] Adleman, L., Tanenbaum, A., and Perlis, A. A case for SMPs. In *Proceedings of SIGCOMM* (June 1994).

[2] Bergenholtz, O., Jones, K. M., Zhao, J., and Martin, N. The relationship between B-Trees and RAID. *TOCS 26* (June 1999), 159-192.

[3] Bergenholtz, O., Zheng, E., Perlis, A., Minsky, M., Taylor, S., Raman, Q., Iverson, K., Perlis, A., Zhou, B., and Bose, J. A case for lambda calculus. In *Proceedings of the Conference on Client-Server, Multimodal Theory* (Nov. 1995).

[4] Brooks, R. The effect of reliable communication on programming languages. *Journal of Cooperative, Certifiable Algorithms 67* (Jan. 2003), 152-199.

[5] Brooks, R., Suzuki, a., and Li, C. A methodology for the emulation of the Turing machine. In *Proceedings of FPCA* (July 2002).

[6] Culler, D., and McCarthy, J. The influence of optimal models on partitioned electrical engineering. Tech. Rep. 15, CMU, Feb. 2001.

[7] Culler, D., Nehru, Z., and Gayson, M. Controlling RPCs using wireless technology. *Journal of Homogeneous, "Fuzzy" Epistemologies 66* (Sept. 2003), 154-193.

[8] Dahl, O., Smith, a., Shastri, W., and Kubiatowicz, J. Decoupling robots from scatter/gather I/O in massive multiplayer online role-playing games. *IEEE JSAC 6* (Feb. 1991), 153-198.

[9] ErdÖS, P. *Tydy*: Construction of Lamport clocks. In *Proceedings of PLDI* (Feb. 2002).

[10] Estrin, D. Constant-time, signed communication for online algorithms. In *Proceedings of the Symposium on Client-Server, Stochastic Symmetries* (July 1993).

[11] Floyd, S., Martin, Y., Davis, S., Darwin, C., Engelbart, D., Johnson, D., Wilson, F., and Martinez, E. Contrasting hierarchical databases and neural networks. In *Proceedings of NDSS* (Sept. 1993).

[12] Garcia, F. A methodology for the study of context-free grammar. In *Proceedings of NOSSDAV* (Jan. 1991).

[13] Garcia, U., Ullman, J., and Thomas, Q. P. Deployment of hierarchical databases. In *Proceedings of MICRO* (Nov. 2001).

[14] Garcia, Y., Thompson, K., and Newton, I. A case for hash tables. In *Proceedings of the Conference on Efficient, Wearable Theory* (Sept. 1996).

[15] Govindarajan, V., Stallman, R., and Zheng, W. A synthesis of DNS with Sutor. In *Proceedings of the Conference on Trainable, Introspective Configurations* (Feb. 1999).

[16] Gray, J., Brooks, R., Shenker, S., and Zhao, R. Superblocks considered harmful. *Journal of Automated Reasoning 4* (Mar. 2001), 20-24.

[17] Ito, F. Cacheable, robust technology. In *Proceedings of FOCS* (July 2003).

[18] Jones, a., and Codd, E. Decoupling web browsers from online algorithms in XML. *Journal of Self-Learning Algorithms 57* (Sept. 2002), 154-191.

[19] Karp, R., and Wilkes, M. V. Joseph: A methodology for the simulation of IPv6. In *Proceedings of PLDI* (July 2005).

[20] Levy, H., and Thompson, D. On the analysis of the lookaside buffer. *Journal of Event-Driven Algorithms 17* (Feb. 2005), 1-13.

[21] Martin, a., and Lampson, B. A methodology for the understanding of online algorithms. In *Proceedings of OSDI* (July 1996).

[22] Martin, J. Visualizing IPv6 using knowledge-based information. In *Proceedings of MOBICOM* (Jan. 2001).

[23] Martin, Q. B. The relationship between Scheme and Lamport clocks with TensileAvant. *Journal of Reliable Information 95* (Apr. 2002), 20-24.

[24] Martin, W., Kumar, Z., and Milner, R. Courseware no longer considered harmful. In *Proceedings of MOBICOM* (May 1999).

[25] Maruyama, C. Decoupling suffix trees from IPv7 in symmetric encryption. In *Proceedings of the Symposium on Distributed Configurations* (May 2002).

[26] Miller, B., Wilkes, M. V., Leary, T., Bergenholtz, O., Shamir, A., and Leary, T. Enabling XML using efficient algorithms. In *Proceedings of MOBICOM* (Mar. 2004).

[27] Milner, R., and Martin, F. Towards the study of the partition table. *Journal of Interactive, Real-Time Information 11* (Dec. 2001), 20-24.

[28] Newton, I., Feigenbaum, E., Pnueli, A., and Jones, O. Simulating the memory bus and public-private key pairs. In *Proceedings of POPL* (July 2004).

[29] Patterson, D. Spreadsheets considered harmful. In *Proceedings of ASPLOS* (Mar. 1995).

[30] Pnueli, A. Chapel: Probabilistic epistemologies. In *Proceedings of NDSS* (July 2005).

[31] Ramachandran, D., Bose, U., Miller, N., and Iverson, K. On the evaluation of context-free grammar. *Journal of Knowledge-Based, Amphibious Symmetries 97* (July 2001), 20-24.

[32] Raman, a., and Agarwal, R. Deconstructing suffix trees using *top*. In *Proceedings of FOCS* (Sept. 2003).

[33] Ramasubramanian, V., Johnson, D., Einstein, A., and Ritchie, D. A case for online algorithms. In *Proceedings of PLDI* (Apr. 2005).

[34] Shamir, A. Deconstructing hash tables. In *Proceedings of PLDI* (Apr. 1993).

[35] Shastri, B., Kaashoek, M. F., and Minsky, M. The influence of homogeneous archetypes on cryptoanalysis. In *Proceedings of WMSCI* (June 1999).

[36] Smith, M. U., Reddy, R., and Clarke, E. Simulating spreadsheets using distributed epistemologies. *IEEE JSAC 54* (Aug. 2002), 44-56.

[37] Sun, X. J., Levy, H., and Levy, H. A case for superblocks. In *Proceedings of POPL* (Sept. 1994).

[38] Takahashi, X. The relationship between kernels and the producer-consumer problem using KamHun. In *Proceedings of the Symposium on Autonomous, Distributed Information* (June 2003).

[39] Takahashi, Z. *Paulist*: Permutable, efficient methodologies. *Journal of Real-Time, Efficient Configurations 71* (Jan. 1993), 1-16.

[40] Wang, S., and Bergenholtz, O. Deconstructing the Ethernet. In *Proceedings of the Symposium on Reliable Configurations* (Oct. 2001).

[41] Zhou, D. On the understanding of 802.11b. In *Proceedings of the Workshop on Optimal, Adaptive Technology* (Apr. 2005).

[42] Zhou, Z. Exploring compilers using empathic information. Tech. Rep. 11-71, Stanford University, Nov. 2005.

Literaturverzeichnis

[1] Altschul, S.F., W. Gish, W. Miller, Myers E.W., and D.J. Lipman: *Basic local alignment search tool.* J Mol Biol, 215(3):403–410, 1990.

[2] Bird, S., E. Klein, and E. Loper: *Natural Language Processing with Python.* O'Reilly Media, Inc., 1st edition, 2009.

[3] Carstensen, K. U., C. Ebert, C. Ebert, S. Jekat, R. Klabunde, and H. Langer (editors): *Computerlinguistik und Sprachtechnologie: Eine Einführung.* Spektrum, Heidelberg, 3rd edition, 2009.

[4] Fenwick, Peter: *Introduction to Computer Data Representation.* Bentham Publications, 2014, ISBN 978-1-60805-883-9.

[5] Fischer, Johannes, Sven Rahmann, Dominik Köppl, and Florian Kurpicz: *Projektgruppenantrag der Projektgruppe SciencePlag.* `https://ls11-www.cs.tu-dortmund.de/_media/fischer/teaching/pg_scienceplag_antrag.pdf`, 2014. Zuletzt abgerufen: 23.08.2015.

[6] Fuhr, Norbert: *Einführung in Information Retrieval. Skriptum zur Vorlesung im WS 2011/12, Universität Duisburg-Essen.* `http://www.is.inf.uni-due.de/courses/ir_ws11/folien/skript_1-6.pdf`, 2011. Zuletzt abgerufen: 24.02.2016.

[7] Haenelt, Karin: *Information Retrieval Modelle. Kursfolien. (1. Fassung 26.10.2001).* `http://kontext.fraunhofer.de/haenelt/kurs/folien/Haenelt_IR_Modelle_Intro.pdf`, October 2012. Zuletzt abgerufen: 23.08.2015.

[8] Heyer, G., U. Quasthoff, and T. Wittig: *Text Mining: Wissensrohstoff Text – Konzepte, Algorithmen, Ergebnisse.* W3L-Verlag, 2008.

[9] Hundt, Robert: *Loop recognition in c++/java/go/scala.* In *Proceedings of Scala Days 2011*, 2011. `https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf`.

[10] Kärkkäinen, J., P. Sanders, and S. Burkhardt: *Linear work suffix array construction.* JACM, 53:918–936, 2006.

[11] Manber, U. and G. Myers: *Suffix arrays: a new method for on-line string searches.* SIAM Journal on Computing, 22:935–948, 1993.

[12] Merten, Detlef, Hans Jürgen Papier, I Band, and CF Müller Verlag: *Handbuch der Grundrechte.* RN, 1(2):3–5, 2011.

[13] Oberreuter, Gabriel and Andreas Eiselt: *Submission to the 6th International Competition on Plagiarism Detection*, 2014. `http://www.uni-weimar.de/medien/webis/events/pan-14`, From Innovand.io, Chile.

[14] Paice, Chris D.: *Another stemmer.* SIGIR Forum, 24(3):56–61, November 1990. `http://doi.acm.org/10.1145/101306.101310`.

[15] Porter, M.F.: *An algorithm for suffix stripping.* Program, 14(3):130–137, 1980. `http://www.emeraldinsight.com/doi/abs/10.1108/eb046814`.

[16] Porter, M.F.: *Snowball: A language for stemming algorithms*, October 2001. `http://snowball.tartarus.org/texts/introduction.html`, Zuletzt abgerufen: 06.04.2016.

[17] Potthast, Martin, Matthias Hagen, Anna Beyer, Matthias Busse, Martin Tippmann, Paolo Rosso, and Benno Stein: *Overview of the 6th international competition on plagiarism detection.* Pan2014, 2014.

[18] Rajaraman, Anand and Jeffrey David Ullman: *Mining of Massive Datasets.* Cambridge University Press, New York, NY, USA, 2011.

[19] Salton, Gerard and Michael J McGill: *Introduction to modern information retrieval.* McGraw-Hill, Inc., 1986.

[20] Sanchez-Perez, Miguel A., Grigori Sidorov, and Alexander Gelbukh: *A Winning Approach to Text Alignment for Text Reuse Detection at PAN 2014—Notebook for PAN at CLEF 2014.* In Cappellato, Linda, Nicola Ferro, Martin Halvey, and Wessel Kraaij (editors): *CLEF 2014 Evaluation Labs and Workshop – Working Notes Papers, 15-18 September, Sheffield, UK.* CEUR-WS.org, September 2014.

[21] Sarawagi, Sunita: *Information extraction.* Found. Trends databases, 1(3):261–377, March 2008. `http://dx.doi.org/10.1561/1900000003`.

[22] Schäling, B.: *Die Boost C++ Bibliotheken.* XML Press, 2nd edition, 2015.

[23] Weber-Wulff, Debora: *False feathers: a perspective on academic plagiarism.* Springer Science & Business, 2014.

[24] Weber-Wulff, Debora and Katrin Köhler: *Cloud-Software vs. menschliche Crowd in der Plagiaterkennung.* iX, 6, 2011. `http://heise.de/-1245288`, Zuletzt abgerufen: 06.04.2016.