Universität Paderborn

Fakultät für Elektrotechnik, Informatik und Mathematik

Institut für Informatik

Fachgebiet Wissensbasierte Systeme

DIPLOMARBEIT

Hashing-basierte Indizierungsverfahren im textbasierten Information-Retrieval

von

B. Sc. Martin Potthast

(martin@potthast.biz)

unter Anleitung von

Prof. Dr. Benno Stein

(stein@upb.de)

Jena, Juni 2006

Danksagung

Diese Diplomarbeit entstand mit einem Aufwand von 188.499 Buchstaben, 21.170 Worten, 1.294 Sätzen, viele davon grammatikalisch korrekt, 35 Formeln mit 66 verschiedenen Symbolen, schätzungsweise 150 Tippfehlern, von denen 95% korrigiert werden konnten, 11 Monaten an ein und demselben Thema, 33 Stunden Diskussion mit dem Betreuer und 2.002 dabei zurückgelegten Kilometern, 3.563 belegten Megabyte, mehr als 100.000 verarbeiteten Testdokumenten, 3 Computern, 8 vernachlässigten Telefonnummern, 5 Reviewern mit insgesamt 736 Anmerkungen, davon 70 in offensichtlich alkoholisiertem Zustand angefertigt, von denen rund 80% umgesetzt wurden, 2 Absätzen, die länger sind als dieser Satz, 50 Umzugskisten, rund 700 heraus- und wieder reingedrehten Schrauben, 3 Umzugshelfern plus 2 Umgezogenen, die kumuliert 15.680 Stufen herauf- und wieder hinabgestiegen sind, 5 zur Entspannung besuchten Musicals, insgesamt aber nur 2 verschiedenen, 70 Litern Schweiß und Tränen, 11 Besuchern, die bis zu 18 Meter Stoff mitbrachten und 2 Kätzchen, die sich darum kümmern werden, 6 Katzenspielzeugen zur Ablenkung davon, 22 Kilo Nudeln, 3 verpassten Geburtstagen, 11 Abenden im Kreise verschiedener Familien, 4 Nächten in einem Gäste-Betonverlies und einem Autor.

Ich danke euch, Stephan, Benno, Martin, Daniel, Carsten, Jens, Lars, Alexander, Daniel, Wiebke und Marc, Mama und Günter, Papa und Hildegard, Roswitha und Herbert, Torsten, Ellinor, Steffi und Olli, denn ohne euch hätten sich die oben genannten Quantitäten so niemals ergeben.

Kurzfassung

Hashing ist ein für die Indizierung und den Ähnlichkeitsvergleich von Objekten bisher wenig beachteter Ansatz. Es ist bekannt, dass mittels Hashing in konstanter Suchzeit festgestellt werden kann, ob eine Menge von Objekten ein bestimmtes Objekt enthält. Die Kollision der Hashwerte zweier Objekte dient dabei als Indikator für ihre Gleichartigkeit. Locality-Sensitive-Hashing verallgemeinert dieses Prinzip. Hier dient die Kollision der Hashwerte zweier Objekte als Indikator für ihre Ähnlichkeit. Damit ist die Extraktion aller zu einem bestimmten Objekt ähnlichen Objekte aus einer Menge in konstanter Zeit möglich.

Abgesehen von Locality-Sensitive-Hashing gibt es nur einen Alternativvorschlag, der auf derselben Idee fußt, nämlich Fuzzy-Fingerprinting. Hier wird ein domänenspezifischer Ansatz zur Indizierung von textuellen Dokumenten verfolgt, der dem domänenunabhängigen Ansatz von Locality-Sensitive-Hashing entgegen steht. Ausgangspunkt und Gegenstand der vorliegenden Ausarbeitung ist die Fragestellung, welcher der beiden Ansätze der bessere zur Indizierung von Dokumenten ist und wie sich die Einbeziehung von Domänenwissen allgemein auf die Indizierung mittels Hashing auswirkt.

Zur Beantwortung dieser Fragen werden beide Ansätze theoretisch aufgearbeitet und in den Kontext der Indizierung von Dokumenten eingebettet. Das Indizierungsprinzip der Transformation wird herausgestellt, das auf der Einbettung indizierter Dokumente in einen anderen Objektraum beruht. Es steht im Gegensatz zum bekannten Prinzip der Inversion, bei dem eine andere Sicht auf die Relation zwischen Dokumenten und ihrem Inhalt erstellt wird. Darüber hinaus werden Konstruktionsprinzipien für Hashfunktionen zur Indizierung von Dokumenten bzw. anderen Objekten identifiziert.

Experimentell zeigt sich, dass das domänenspezifische Fuzzy-Fingerprinting dem uninformierten Locality-Sensitive-Hashing bei der Indizierung von Dokumenten überlegen ist. Dies äußert sich insbesondere in einem höheren durchschnittlichen Recall einer Suchanfrage auf dem Index. Es zeigt sich also, dass die Ausnutzung von Domänenwissen bei der Konstruktion von Hashfunktionen größeren Erfolg verspricht als eine domänenunabhängige Herangehensweise.

Die Ausarbeitung unterteilt sich in fünf Kapitel und einen Anhang. Im ersten Kapitel werden grundlegende Begriffe der Indizierung im Kontext des textbasierten Information-Retrieval aufgearbeitet sowie die Indizierungsprinzipien Transformation und Inversion erläutert. Im zweiten Kapitel werden aktuelle Indizierungsansätze beschrieben und verglichen und im dritten Kapitel die oben erwähnten hashing-basierten Indizierungsverfahren im Detail erläutert. Das vierte Kapitel beschreibt die Ergebnisse der durchgeführten Experimente. Das fünfte Kapitel fasst die Beiträge der Arbeit zusammen und gibt Ausblick auf kommende Studien. Der Anhang vertieft die für die Experimente benötigte Technik der Clusteranalyse.

Inhaltsverzeichnis

Kapitel 1

Prinzipien der Indizierung

Das World-Wide-Web ist die bis dato größte „Bibliothek“, die dem Menschen je zur Verfügung stand. Zum Bestand gehören hauptsächlich Dokumente in Form von Text, Bild, Ton oder Video. Mit ihrer Hilfe ist es möglich, den Informationsbedarf vieler Personen zu befriedigen. Mit Informationsbedarf ist dabei ein Mangel an Wissen gemeint, der eine Person davon abhält, eine Aufgabe zu erfüllen. Im Fachgebiet Information-Retrieval werden Technologien erforscht, die bei der Beschaffung und Auswahl geeigneter Quellen unterstützen sollen, um einen Wissensmangel zu befriedigen. Am Anfang steht hier die Formulierung einer Frage, auf die unter allen zur Verfügung stehenden Dokumenten nach Antworten gesucht wird. In der Tat ist es die Suche selbst, die einen der Hauptaspekte bei der Benutzung des Webs ausmacht. Der Grund dafür liegt in der Fülle der zur Verfügung stehenden Dokumente, da häufig selbst zu schwierigen Fragestellungen eine Vielzahl von Quellen existieren. Das begründet auch den Erfolg, den die Anbieter von Suchmaschinen und Suchtechnologie verzeichnen.

Daran anknüpfend stellt sich die Frage, wie das Internet effizient durchsucht werden kann, denn eine sequentielle Suche auf allen Dokumenten ist zu aufwändig. Der Bestand von Bibliotheken wird zu diesem Zweck schon seit Urzeiten nach verschiedenen Kriterien sortiert und katalogisiert. Dieser Vorgang heißt Indizierung und sein Ergebnis Index. Dabei handelt es sich um eine Datenstruktur, die beispielsweise Stichworte Dokumenten zuordnet. Die Suche nach Informationen verlagert sich so von den Dokumenten auf den Index. Die Effizienz der Suche wird damit gesteigert, da anstatt aller Dokumente nur noch der Index und im Anschluss daran alle potenziell relevanten Dokumente durchsucht werden müssen. Das setzt natürlich voraus, dass ein Suchender seinen Informationsbedarf hinreichend gut formuliert, so dass ein passender Eintrag im Index gefunden werden kann.

Die Anforderungen an einen Index für das Internet sind enorm. Er muss Milliarden von Dokumenten indizieren und in Bruchteilen von Sekunden durchsuchbar sein. Darüber hinaus müssen neue Dokumente schnell in den Index aufgenommen und veraltete schnell gelöscht werden kön-

Abbildung 1.1: Die Suche nach einem Leerzeichen bzw. dem regulären Ausdruck * * ist ein Indiz für die Größe des Indexes von Google. Die zurückgelieferten Werte schwanken allerdings erheblich zwischen 8 und 25 Milliarden Dokumenten, je nachdem von welcher Länder-Domain aus gesucht wird. Die unterschiedlichen Zahlen hängen vermutlich davon ab, welches der verschiedenen Datenzentren Googles angesprochen wird.

nen. Die Leistungsfähigkeit von Indizes lässt sich am besten am Beispiel großer Suchmaschinen aufzeigen, offizielle Angaben zu Indexgrößen sind jedoch rar. Einen Eindruck der Größe von Googles Index vermittelt das Ergebnis der Suche nach einem Zeichen, das auf nahezu jeder Webseite vorkommt, dem Leerzeichen. Zwar ist die in Abbildung 1.1 zu sehende Zahl von 25 Milliarden Webdokumenten höchstwahrscheinlich sehr ungenau, doch die Größenordnung, in der sie sich bewegt, ist zutreffend.

Es existieren zahlreiche Datenstrukturen, die als Index Verwendung finden. Welche davon für eine konkrete Aufgabenstellung am besten geeignet ist, hängt von der Art der zu erwartenden Fragen ab, also wie ein Suchender seinen Informationsbedarf formuliert. Beides hängt wiederum mit den Repräsentationsformen von Dokumenten zusammen. Grundlegende Prinzipien der Indizierung und der Zusammenhang zwischen Dokumentmodell, Art der Frage und Indizierungsverfahren werden in den folgenden Abschnitten eingeführt. Im nächsten Kapitel folgt ein detaillierter Überblick über den Stand der Forschung zu Indizierungsverfahren. Wir schlagen hier die Unterscheidung zwischen zwei grundlegend verschiedenen Konzepten der Indizierung vor, der Inversion und der Transformation. In den Kapiteln 3 und 4 sind zwei Vertreter der transformativen Indizierung, Locality-Sensitive-Hashing und Fuzzy-Fingerprinting, Gegenstand vergleichender Analysen und Experimente.

Zunächst jedoch wird der zentrale Begriff „Dokument“ genauer definiert.

1.1 Der Begriff „Dokument“

Ein Dokument d ist Baeza-Yates und Ribeiro-Neto (1999) zufolge ein Text, der entweder selbst eine *logische Einheit* bildet oder Teil einer solchen ist. Als logische Einheit werden Objekte bezeichnet wie sie in der Realität auftreten, zum Beispiel ein Buch, eine Webseite, eine Textdatei, ein beschriebenes Papier usw.

Hervorzuheben ist hier die Unterscheidung zwischen Dokument und logischer Einheit, denn es wäre opportun die beiden Begriffe synonym zu verwenden. Zur Effizienzsteigerung wer-

den jedoch häufig nur Ausschnitte aus logischen Einheiten verarbeitet. Bei der Interaktion mit Suchmaschinen werden zum Beispiel die gelieferten Webseitenausschnitte um die gefundenen Suchbegriffe als Dokumente betrachtet, anstatt die verlinkten Seiten vollständig herunterzuladen. Die Unterscheidung der beiden Begriffe ermöglicht es also, in theoretischen Betrachtungen vom generischen Startpunkt eines Dokuments auszugehen.

1.2 Dokumentmodelle

Digitale Dokumente werden in verschiedenen Formaten gespeichert oder ihr Inhalt mithilfe von Auszeichnungssprachen strukturiert. Letztere sind — grob gesagt — all jene Sprachen, deren Akronym auf „-ML“ für „Markup Language“ endet. Da verschiedene Formate und Auszeichnungssprachen zugleich auftreten können, wird auch davon abstrahiert und eine *logische Sicht* auf ein Dokument etabliert. Ziel dabei ist die Definition eines (Dokument-)Modells $\mathbf{d}$ zur Repräsentation von d.

Die einfachste Form eines Modells ist eine Zeichenkette. Eine Folge von Symbolen, die einem endlichen Alphabet entnommen sind, repräsentiert hier das Dokument. Der Umfang des Alphabets, mit dem modelliert wird, bedingt den Grad der Abstraktion des Modells gegenüber dem Dokument.

Das bekannteste Dokumentmodell im Information-Retrieval ist das Vektorraummodell nach Salton und Lesk (1968). Ein Dokument d einer Dokumentmenge D wird hier durch einen m-dimensionalen Vektor $\mathbf{d}$ modelliert. Jede Dimension des dadurch aufgespannten Vektorraums steht stellvertretend für ein Wort w des Vokabulars W, der Menge aller Worte, die in der Dokumentmenge verwendet werden. Es gilt daher $m = |W|$. Die i-te Vektorkomponente des Vektors $\mathbf{d}$ beziffert die Wichtigkeit des i-ten Wortes aus W in Bezug auf den Inhalt des Dokuments d.

Um die Wichtigkeit eines Wortes in Bezug auf den Inhalt eines Dokuments zu messen, werden so genannte Termgewichtsmaße eingesetzt. Ein weit verbreitetes Maß hierfür ist zum Beispiel $tf \cdot idf$, das eine Kombination aus der Häufigkeit (tf) eines Wortes w im Dokument d und der inversen Dokumenthäufigkeit (idf) ist. Letztere errechnet sich zum Beispiel durch $idf(w) = \log(n/df(w))$, wobei $df(w)$ die Anzahl der Dokumente ist, die w enthalten, und n die Größe der Dokumentmenge. Worte, die in wenigen Dokumenten häufig enthalten sind, werden hoch bewertet, da sie gut zur Abgrenzung der betroffenen Dokumente von anderen geeignet sind.

Es ist zu bemerken, dass im Vektorraummodell keinerlei Informationen über die Reihenfolge der Worte im Dokument modelliert werden. Diese Informationen sind jedoch für die Beantwortung mancher Formen des Informationsbedarfs unerlässlich. Mithilfe des Suffixbaummodells nach Meyer zu Eissen u. a. (2005) können Informationen über die Wortreihenfolge im Modell bewahrt werden. Ein Dokument wird hier durch einen Suffixbaum[1] repräsentiert, der alle im

Dokument enthaltenen (Teil-)Folgen von Worten indiziert.

Während der Modellbildung von Dokumenten werden häufig so genannte Stoppworte ausgeschlossen. Dabei handelt es sich um Worte, die nur einen geringen Beitrag zur Beschreibung des Inhalts leisten und in erster Linie dem Satzgefüge dienen. Worte, die im Sprachgebrauch besonders häufig verwendet werden und in den meisten Dokumenten zu finden sind, zählen auch dazu. Eine darüber hinaus oft angewandte Technik ist Stemming, womit die Abbildung verschiedener Flexionen gleichbedeutender Worte auf eine gemeinsame Stammform gemeint ist. Eingesetzt werden hierfür sprachabhängige, regelbasierte Algorithmen, wie zum Beispiel Porters Stemming-Algorithmus (Porter 1980), oder sprachunabhängige, statistische Verfahren, wie Successor-Variety-Stemming (Stein und Potthast 2006).

1.3 Formulierung eines Informationsbedarfs als Frage

Ein Informationsbedarfs q führt bei der Verwendung eines Informationssystems, zum Beispiel einer Suchmaschine, zur Formulierung einer Frage $\mathbf{q}$ an das System. Es ist jedoch nicht jede Frage zulässig. Abhängig vom Abstraktionsgrad des zugrundeliegenden Dokumentmodells gibt es Fragen, die nicht beantwortet werden können. Das Vektorraummodell erlaubt zum Beispiel nicht die Beantwortung von Fragen, die sich auf die Reihenfolge von Worten beziehen, da es diese Information nicht modelliert.

Grundlegend unterscheiden wir drei Typen von Fragen[2]:

- Wortfragen: Die bekannteste Form der Frage ist das Stichwort. Es handelt sich dabei um ein Wort w aus dem Vokabular W, das dazu geeignet erscheint, das Gesuchte genau zu „treffen“. Als relevant werden all jene Dokumente betrachtet, die w enthalten.

- Kontextfragen: Dieser Fragetyp teilt sich weiter auf in Fragen nach Nachbarschaft von Worten und Fragen nach Phrasen. Für eine Menge bzw. Folge von Worten werden alle Dokumente gesucht, in denen diese Worte nahe zusammen bzw. als Phrase auftreten. Ziel ist es, die Suche zu verfeinern, so dass nur noch Dokumente als relevant betrachtet werden, in denen ein Wort im Kontext (in der Nähe) anderer Worte verwendet wird.

- Ähnlichkeitsfragen: Wenn bereits ein Dokument d bekannt ist, dessen Inhalt sich auf das

[1] Die Datenstruktur Suffixbaum wird im Rahmen der Arbeit an verschiedenen Stellen verwendet. An dieser Stelle genügt eine kurze Erläuterung: Für ein Dokument $d = w_1 ... w_{|d|}$ ist das i-te Suffix die Teilfolge $w_i ... w_{|d|}$. Ein Suffixbaum für d wird wie folgt konstruiert: Das i-te Suffix von d wird in den Baum eingefügt, indem geprüft wird, ob von der Wurzel eine Kante ausgeht, die mit w_i beschriftet ist. Wenn das der Fall ist, wird sie traversiert und analog beim Nachfolgeknoten mit dem Wort w_{i+1} verfahren, und so weiter. Sollte unterdessen ein Knoten erreicht werden, der keine passende Ausgangskante hat, so wird ihm ein neuer Knoten als Kind hinzugefügt und die Kante mit dem aktuellen Wort beschriftet.

Gesuchte bezieht, lässt es sich als Beispiel dafür verwenden. Relevant sind daher alle Dokumente d' einer Menge von Dokumenten D, die inhaltlich ähnlich zu d sind.

Übergeordnet sind so genannte boolesche Fragen. Es handelt sich dabei um logische Kombinationen von Fragen eines Typs. Eingesetzt werden im Allgemeinen die Junktoren $\wedge$, $\vee$ und $\neg$, wobei Fragen als Atome aufgefasst werden, deren Wahrheitswert 1 ist, falls es wenigstens ein Antwortdokument gibt. Die effiziente Beantwortung einer booleschen Frage basiert auf der Vereinfachung ihres logischen Ausdrucks durch Transformation in die konjunktive Normalform und der geschickten Auswahl der Klausel, für die zuerst die Antwortmenge berechnet wird. Nach Witten u. a. (1999) sollte diejenige Klausel zuerst bearbeitet werden, deren Antwortmenge am kleinsten ist. Dadurch wird die Anzahl der potenziell für die gesamte boolesche Frage relevanten Dokumente von vornherein minimiert. Da a-priori nicht bekannt ist, auf welche Klausel am wenigsten Dokumente zutreffen, kommen Auswahlheuristiken zum Einsatz. Beispielsweise könnte immer zuerst die Klausel ausgewertet werden, in der nach Worten gefragt wird, die nur selten vorkommen.

1.4 Befriedigung eines Informationsbedarfs

Die Befriedigung eines Informationsbedarfs geschieht mithilfe der oben vorgestellten Dokumentmodelle und Fragetypen. Sei $\mathcal{R}$ das Konzept einer Relation zwischen einer Menge von Dokumenten D und einer Menge spezifischer Informationsbedarfe Q. Wenn ein Dokument $d \in D$ sich zur Befriedigung eines Informationsbedarfs $q \in Q$ eignet, dann gilt $(d, q) \in \mathcal{R}$, andernfalls $(d, q) \notin \mathcal{R}$.

Zur Prüfung dieses Sachverhalts für ein konkretes d und q werden stellvertretend das Dokumentmodell $\mathbf{d}$ aufgestellt, die Frage $\mathbf{q}$ formuliert und eine konkrete Relation R definiert, die das konzeptuelle $\mathcal{R}$ widerspiegelt. Es wird geprüft, ob $(\mathbf{d}, \mathbf{q}) \in R$ gilt, und anschließend das Ergebnis als Antwort auf die Ausgangsfrage interpretiert. Abbildung 1.2 illustriert diesen Vorgang. Dabei wird *angenommen*, dass ein Rückschluss auf das Verhältnis zwischen d, q und $\mathcal{R}$ möglich ist. Ob die Annahme zutrifft, hängt davon ab, wie gut das Dokumentmodell und die Frage das Dokument bzw. den Informationsbedarf repräsentieren. Ob das Prozedere effizient ist, hängt dagegen von der Stärke der Abstraktion ab. Es gilt bei der Modellbildung für Dokumente und der Formulierung von Fragen also, Güte gegen Effizienz abzuwägen.

Für die drei vorgestellten Fragetypen werden verschiedenartige Relationen R zwischen Dokumentmodellen und Fragen benötigt. Bei Wort- oder Kontextfragen muss das Enthaltensein eines bzw. mehrerer Worte im Dokument geprüft werden. Bei Kontextfragen muss darüber hin-

[2] Darüber hinaus können noch weitere Typen identifiziert werden. Für die im Rahmen dieser Arbeit behandelten Fragestellungen genügt diese Unterscheidung jedoch.

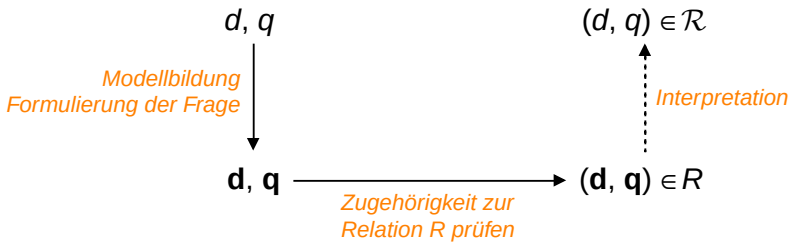

Abbildung 1.2: Die Überprüfung, ob ein Dokument d sich zur Befriedigung eines Informationsbedarfs q eignet, verläuft in drei Schritten: 1. Das Dokumentmodell $\mathbf{d}$ wird aufgestellt und die Frage $\mathbf{q}$ formuliert. 2. Es wird geprüft, ob $(\mathbf{d}, \mathbf{q})$ teil von R sind 3. Das Ergebnis wird als Antwort auf die Ausgangsfrage interpretiert.

aus die Bedingung erfüllt sein, dass die Worte nahe genug beieinander oder in einer bestimmten Reihenfolge auftreten.

Für Ähnlichkeitsfragen kann R zwischen Dokumentmodellen und Fragen als „hinreichende Ähnlichkeit“ oder „hinreichende Distanz“ definiert werden. Es ist zu berücksichtigen, dass eine Ähnlichkeitsfrage $\mathbf{q}$ selbst ein Dokumentmodell ist, weshalb eine Funktion notwendig wird, die die Ähnlichkeit (Distanz) zwischen zwei Dokumentmodellen berechnen kann. „Hinreichend“ meint in diesem Zusammenhang, dass ein Dokument nur dann zur Befriedigung eines Informationsbedarfs geeignet ist, falls ein gewisser Ähnlichkeitsschwellwert ε zwischen den korrespondierenden Dokumentmodellen nicht unterschritten (überschritten) wird. Nachfolgend werden verschiedene Funktionen zur Berechnung der Ähnlichkeit bzw. Distanz zwischen Dokumentmodellen vorgestellt.

1.5 Ähnlichkeits- und Distanzfunktionen

Tan u. a. (2005) definieren die Ähnlichkeit zweier Dokumente anhand ihrer Modelle $\mathbf{d}_1$ und $\mathbf{d}_2$ allgemein als einen numerischen Wert aus dem Intervall $[0, 1]$, wobei ein Betrag nahe 0 geringe und ein Betrag nahe 1 große Ähnlichkeit bedeutet. Eine Funktion $\varphi(\mathbf{d}_1, \mathbf{d}_2)$ heißt Ähnlichkeitsfunktion, wenn sie solche Werte berechnet. Die Distanz zwischen den Dokumenten ist analog definiert als Wert aus $[0, 1]$ oder aber aus $[0, \infty)$, wobei 0 keine und 1 bzw. ∞ maximale Distanz bedeutet. Eine Funktion $\sigma(\mathbf{d}_1, \mathbf{d}_2)$ heißt Distanzfunktion, wenn sie solche Werte berechnet.

Zwei Ähnlichkeitsfunktionen, die im Information-Retrieval oft verwendet werden, sind der Jaccard-Koeffizient und die Kosinusähnlichkeit. Beide haben die Eigenschaft, dass sie den Ähnlichkeitswert für zwei Dokumente allein auf Basis ihrer Gemeinsamkeiten errechnen. Wir unterscheiden in diesem Zusammenhang zwei Formen der Gemeinsamkeit, die Überlappung und den Ausschluss. Für ein beliebiges Wort w liegt überlappende Ähnlichkeit zwischen Dokumenten vor, wenn beide w enthalten, und umgekehrt ausschließende Ähnlichkeit, wenn beide w nicht enthalten. Ausschließende Gemeinsamkeiten bleiben in den Ähnlichkeitsfunktionen je-

doch unberücksichtigt, da sie nur einen geringen Beitrag zur Ähnlichkeit der Dokumente leisten können. Wenn also ein Dokument w enthält, behandelt es einen Themenbereich, der mit w umschrieben werden kann. Andernfalls ist nur der Schluss zulässig, dass das Dokument diesen Themenbereich nicht behandelt. Wenn also keins der beiden Dokumente w enthält, dann ist auf dieser Grundlage nur der Ausschluss eines Themenbereichs möglich, der von beiden nicht tangiert wird. Das bedeutet jedoch nicht, dass sich die Dokumente in anderen Themenbereichen überschneiden, was jedoch die Grundvoraussetzung für einen Ähnlichkeitswert ungleich 0 ist.

Der Jaccard-Koeffizient für Dokumente basiert auf dem Vektorraummodell mit einem binären Termgewichtsmaß[3]. Er lässt sich unter Zuhilfenahme des Skalarprodukts $\mathbf{d}_1^T\mathbf{d}_2$ und der L_1-Norm $\|\mathbf{d}_i\|_1$ wie folgt definieren:

$$\varphi_J(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1^T\mathbf{d}_2}{\|\mathbf{d}_1\|_1 + \|\mathbf{d}_2\|_1 - \mathbf{d}_1^T\mathbf{d}_2}$$

Die Kosinusähnlichkeit für Dokumente basiert auf dem Vektorraummodell mit einem beliebigen Termgewichtsmaß. Sie erlaubt damit die Berechnung eines Ähnlichkeitswertes anhand von Dokumentvektoren, in denen komplexere als das binäre Termgewichtsmaß verwendet werden. Der Name „Kosinusähnlichkeit“ leitet sich von dem berechneten Wert der Funktion ab; sie errechnet den Kosinus des Winkels zwischen zwei Dokumentvektoren. Je spitzer der Winkel ist, desto größer ist die rechnerische Ähnlichkeit. Die Kosinusähnlichkeit ist analog zum Jaccard-Koeffizienten unter Verwendung des Skalarprodukts und der L_2-Norm $\|\mathbf{d}_i\|_2$ wie folgt definiert:

$$\varphi_{\cos}(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1^T\mathbf{d}_2}{\|\mathbf{d}_1\|_2 \cdot \|\mathbf{d}_2\|_2}$$

Die L_1-Norm und die L_2-Norm sind weithin bekannt und können auch zur Bestimmung von Distanzen verwendet werden, wenn die beteiligten Dokumentvektoren zuvor komponentenweise voneinander subtrahiert werden. Es wird in diesem Fall auch von der Manhatten- bzw. euklidischen Distanz gesprochen. Dabei handelt es sich um Spezialisierungen der Minkowski-Distanz mit dem Wertebereich $[0, \infty)$, die für zwei m-dimensionale Vektoren $\mathbf{d}_1^T = (x_1, \ldots, x_m)$ und $\mathbf{d}_2^T = (y_1, \ldots, y_m)$ und eine Konstante c wie folgt definiert ist:

$$\sigma_M(\mathbf{d}_1, \mathbf{d}_2) = \left(\sum_{i=1}^{m} |x_i - y_i|^c \right)^{1/c}$$

Die Algorithmen und Verfahren, die in späteren Abschnitten vorgestellt und verwendet werden,

[3] Ein binäres Termgewichtsmaß liegt vor, wenn die Werte aller Vektorkomponenten eines Dokumentvektors aus $\{0, 1\}$ stammen, wobei 1 bedeutet, dass das durch die Komponente repräsentierte Wort im Dokument enthalten ist, und 0, dass das nicht der Fall ist.

basieren einerseits auf Ähnlichkeits- und andererseits auf Distanzfunktionen. Da aber alle Erwägungen im Kontext des textbasierten Information-Retrieval stattfinden, in dem im Normalfall von einer Ähnlichkeitsfunktion ausgegangen wird, ist eine Transformation von Ähnlichkeits- und Distanzwerten nötig. Die Funktion $\sigma_\varphi(\mathbf{d}_1, \mathbf{d}_2)$ sei definiert als Distanzfunktion mit Wertebereich $[0, 1]$, die auf Basis einer Ähnlichkeitsfunktion $\varphi(\mathbf{d}_1, \mathbf{d}_2)$ mit demselben Wertebereich wie folgt berechnet wird:

$$\sigma_\varphi(\mathbf{d}_1, \mathbf{d}_2) = 1 - \varphi(\mathbf{d}_1, \mathbf{d}_2)$$

1.6 Ansätze zur Indizierung

Wir unterscheiden bei allen heute bekannten Indizierungsverfahren zwischen zwei grundlegend verschiedenen Ansätzen, der Inversion und der Transformation. Obgleich Vertreter beider Ansätze schon sehr lange bekannt sind und verwendet werden, ist bisher nur das Prinzip der Inversion als solches erkannt worden. Die Ursache dafür liegt wahrscheinlich darin, dass es lange Zeit nur einen Vertreter der Transformation gab, die Signaturliste, und erst jüngste Entwicklungen ein zweites Verfahren hervorgebracht haben, nämlich Similarity-Hashing.

Das Prinzip inverser Indizierung ist jedem bekannt, der schon einmal einen Stichwortindex eines Fachbuches verwendet hat. Allgemein existiert zwischen Dokumenten und Worten eine $n : |d|$-Relation. Jedes Dokument kann also bis zu $|d|$ Worte enthalten, und jedes Wort in bis zu n Dokumenten enthalten sein. Eine Sicht auf die Relation, nämlich die Abbildung von Dokumenten auf Worte ist explizit durch die Dokumentmenge D gegeben. Es wird jedoch die umgekehrte Abbildung zur Beantwortung von Fragen benötigt. Inversion bedeutet daher das Explizitmachen der Abbildung von Worten auf Dokumente, also die Aufstellung einer alternativen Sicht auf die Relation. Der bekannteste Vertreter der Inversion ist die invertierte Liste.

Die transformative Indizierung basiert auf einem ähnlichen Prinzip wie die Modellbildung für Dokumente (siehe Abschnitt 1.1 und Abbildung 1.3). Seien ein Dokumentmodell $\mathbf{d}$, eine Frage $\mathbf{q}$ und eine Relation R gegeben. Zu klären ist, ob $(\mathbf{d}, \mathbf{q}) \in R$ gilt. Zu diesem Zweck werden sowohl $\mathbf{d}$ als auch $\mathbf{q}$ mithilfe einer Transformationsvorschrift h in einen Objektraum eingebettet und darin durch die Objekte $h(\mathbf{d})$ und $h(\mathbf{q})$ repräsentiert. Auf dem Objektraum sei eine Relation R_h definiert. Es wird geprüft, ob $(h(\mathbf{d}), h(\mathbf{q})) \in R_h$ gilt und anschließend das Ergebnis als Antwort auf die Ausgangsfrage interpretiert. Analog zur Modellbildung für Dokumente wird auch hier *angenommen*, dass ein Rückschluss von R_h auf R möglich ist. Die Richtigkeit der Annahme ist davon abhängig, ob die Transformationsvorschrift h in Zusammenhang mit R_h die Relation zwischen $\mathbf{d}$ und $\mathbf{q}$ hinreichend gut widerspiegelt. Die konkreten Ausprägungen von R und R_h hängen von den zu beantwortenden Fragen und der Transformationsvorschrift ab.

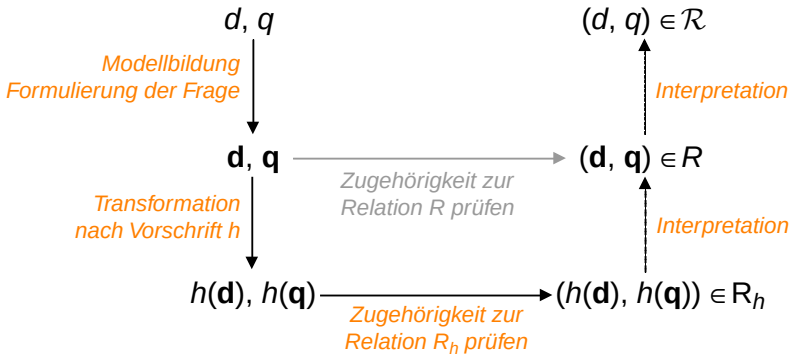

Abbildung 1.3: Die Überprüfung, ob eine Relation R zwischen einem Dokumentmodell $\mathbf{d}$ und einer Frage $\mathbf{q}$ gilt, verläuft in drei Schritten: 1. Mit einer Transformationsvorschrift h werden die Objekte $h(\mathbf{d})$ und $h(\mathbf{q})$ erzeugt. 2. Es wird geprüft, ob zwischen beiden Objekten die Relation R_h gilt. 3. Das Ergebnis wird als Antwort auf die Ausgangsfrage interpretiert.

Die oben und in Abschnitt 1.3 vorgestellte Differenzierung zwischen Indizierungsverfahren bzw. Fragetypen ist in Abbildung 1.4 zusammengefasst. Je nach Fragetyp und Indizierungsansatz kommen verschiedene Indizierungsverfahren infrage. Die in der Abbildung angegebenen Technologiebeispiele werden im folgenden Kapitel näher betrachtet.

1.7 Bewertung von Indizierungsverfahren

Da es zur Beantwortung jedes Fragetyps mehrere mögliche Indizierungsverfahren gibt, bleibt zu klären, anhand welcher Kriterien sie untereinander vergleichbar sind. Es werden sowohl Retrieval-Eigenschaften als auch algorithmische Kriterien berücksichtigt. Zu den algorithmischen zählen die Geschwindigkeit der Suche auf dem Index und der zusätzlich zur Speicherung der Dokumente anfallende Platzverbrauch. Als Referenzgröße dient hier die Anzahl der zu indizierenden Dokumente n. Insbesondere der Platzverbrauch wird in jedem Index stark von der Genauigkeit der Indizierung beeinflusst, der so genannten Granularität des Indexes. Je genauer Dokumente indiziert werden, desto größer wird auch ihr Index.

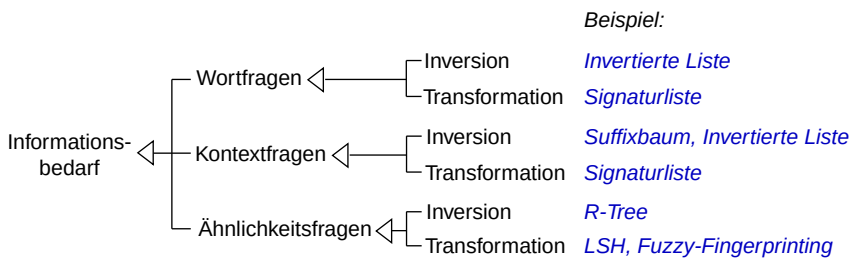

Abbildung 1.4: Eine Taxonomie von Indizierungsverfahren für die effiziente Beantwortung von Fragen. Ein Informationsbedarf wird mithilfe eines der Fragetypen formuliert. Für einige Fragetypen gibt es Indizierungsverfahren, die sich jeweils in inverse und transformative Indizierung unterteilen.

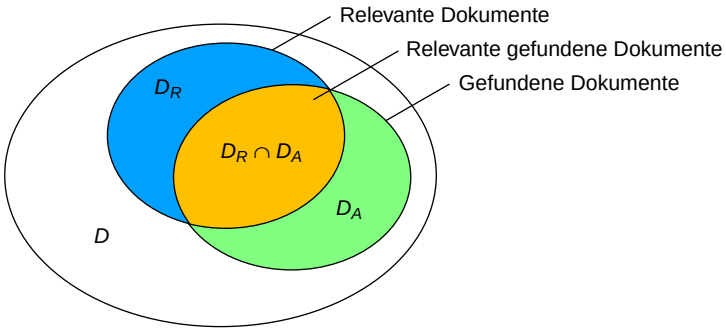

Abbildung 1.5: Für eine beliebige Suchanfrage auf einer Dokumentmenge D misst die Precision den Anteil von $D_R \cap D_A$ an D_A und der Recall den Anteil von $D_R \cap D_A$ an D_R.

Inverse Indizierungsverfahren ermöglichen meist die exakte Beantwortung von Fragen, also das Auffinden aller für eine Frage relevanten Dokumente aus einer gegebenen Dokumentmenge. Es gibt jedoch auch Verfahren, die nur eine inexakte Antwort zulassen, so dass auch irrelevante Dokumente bzw. nicht alle relevanten Dokumente gefunden werden. In diesem Zusammenhang wird von falsch positiv bzw. falsch negativ deklarierten Dokumenten gesprochen. Die Bewertung der Qualität der Indizierung spielt hier also eine besondere Rolle. Ein Qualitätsverlust ist oft nur durch einen erheblichen algorithmischen Vorteil gegenüber anderen Verfahren zu rechtfertigen. In vielen Anwendungsfällen genügt die unscharfe Beantwortung von Fragen, wie zum Beispiel bei der Internetsuche.

Für die Bewertung der Qualität lassen sich die Retrieval-Eigenschaften Precision und Recall messen. Sie setzen die Menge aller relevanten Dokumente $D_R \subseteq D$ mit der Menge gefundener Dokumente $D_A \subseteq D$ auf unterschiedliche Weise ins Verhältnis. Unter der Annahme, dass weder D_R noch D_A leer sind, errechnen sie sich wie folgt:

$$prec = \frac{|D_R \cap D_A|}{|D_A|}$$
$$rec = \frac{|D_R \cap D_A|}{|D_R|}$$

Es ist leicht zu erkennen, das von den Retrieval-Eigenschaften her „perfekte“ Indizierungsverfahren eine Precision und einen Recall von jeweils 1 erreichen. Buckland und Gay (1994) haben die Beziehung zwischen Precision und Recall bei verschiedenen inexakten Verfahren des Information-Retrieval untersucht und stellten dabei insbesondere die inverse Beziehung der beiden Größen heraus. Ein hoher Recall bedeutet zumeist niedrige Precision und umgekehrt. Es ist daher oft notwendig, einen Kompromiss zwischen den beiden Größen zu erlangen. Abbildung 1.6 illustriert dies durch die Darstellung der Precision in Abhängigkeit vom Recall. Im Information-Retrieval werden häufig Kurven dieser Form erzielt.

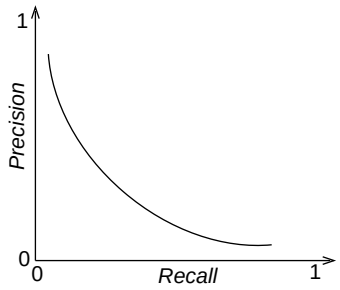

Abbildung 1.6: Darstellung der inversen Beziehung zwischen Precision und Recall. Hoher Recall bedeutet meist niedrige Precision und umgekehrt.

Kapitel 2

Indizierungsverfahren

Die Indizierung großer Mengen von Dokumenten erfordert Sorgfalt bei der Auswahl von Datenstrukturen und Algorithmen. Heutige Indizierungsverfahren sind komplexe Kombinationen verschiedener Datenstrukturen, Kompressionsalgorithmen und Algorithmen für den parallelen Zugriff. In den folgenden Abschnitten werden Vertreter inverser und transformativer Indizierung in ihren Grundzügen erläutert und ein Überblick über den Stand der Forschung zum jeweiligen Verfahren gegeben. Die nachstehend aufgeführten Punkte dienen dabei als Richtschnur bei der Betrachtung:

- Datenstruktur: Der Aufbau des Indexes im Speicher.
- Suche: Algorithmen zur Abarbeitung aller beantwortbaren Fragetypen.
- Konstruktion: Algorithmen zur Konstruktion des Indexes für eine gegebene Dokumentmenge.
- Bibliographie: Überblick über aktuelle Forschungen zum betrachteten Indizierungsverfahren.

Wo immer möglich werden die Verfahren anhand eines einfachen Beispiels illustriert. Es handelt sich um einen Vierzeiler, bei dem jeder Vers als Dokument betrachtet wird. Die beiden letzten Verse teilen je mit dem ersten ein Wort, so dass nichttriviale Beispiele entstehen. Der Text lautet:

> Ein Naßhorn und ein Trockenhorn
> spazierten durch die Wüste,
> da stolperte das Trockenhorn,
> unds Naßhorn sagte: „Siehste !“

Heinz Erhardt (1906–1979)

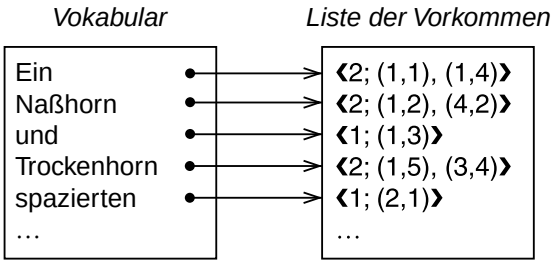

Abbildung 2.1: Eine invertierte Liste für eine Menge von Dokumenten. Das Vokabular enthält alle Worte des Textes in einer geeigneten Datenstruktur. Von jedem Wort w wird ein Eintrag in der Liste der Vorkommen referenziert. Jeder Eintrag enthält ein Array der Form $\langle f(w); (i,j), \ldots \rangle$, in dem $f(w)$ die Worthäufigkeit ist und das Tupel (i,j) besagt, dass w in Dokument d_i an Position j steht. Alternativ kann die Angabe der Position im Dokument auch entfallen.

2.1 Inverse Indizierung

Der bekannteste Vertreter inverser Indizierung ist die invertierte Liste. Es eignet sich zur Beantwortung von Wort- und Kontextfragen. Für Kontextfragen wird der Suffixbaum und für Ähnlichkeitsfragen der R-Tree vorgestellt.

2.1.1 Invertierte Liste

Eine invertierte Liste ist nach Baeza-Yates und Ribeiro-Neto (1999) unterteilt in das Vokabular und die Liste der Vorkommen. Das Vokabular speichert alle bekannten Worte W. Der i-te Eintrag in der Liste der Vorkommen enthält Referenzen auf die Teilmenge $\mathbf{D}_i$ der Dokumente $\mathbf{D}$, die jeweils das i-te Wort des Vokabulars enthalten[1]. Abhängig von der Granularität der Indizierung enthält die Liste der Vorkommen zusätzlich zu den Referenzen Angaben über die Position(en) des Wortes im Dokument. Abbildung 2.1 zeigt den schematischen Aufbau einer invertierten Liste.

Als Datenstruktur für das Vokabular wird häufig ein Patricia-Trie[2] eingesetzt. Dabei handelt es sich um einen digitalen Suchbaum, in dem Pfade, die sich nicht verzweigen, zu einem Knoten zusammengefasst werden. In einem solchen Knoten werden die Anzahl der Zeichen gespeichert, die bei der Suche im Baum in der gesuchten Zeichenfolge übersprungen werden können. Vergleichbar ist diese Datenstruktur mit der des Suffixbaums, da sie analog konstruiert wird. Die Blätter des Baumes referenzieren je einen Eintrag in der Liste der Vorkommen.

Die Liste der Vorkommen besteht aus $|W|$ Einträgen. Jeder Eintrag ist ein Array von Referenzen

[1] Die Unterscheidung von Dokumenten und Dokumentmodellen wird an dieser Stelle und in den Folgekapiteln fallen gelassen. Deshalb ist hier von Dokumenten die Rede, obwohl tatsächlich Dokumentmodelle gemeint sind. Alle weiteren Erwägungen gründen darauf, dass für ein zu verarbeitendes Dokument *immer* zunächst ein geeignetes Dokumentmodell aufgestellt wird. Beide Begriffe werden daher gleichbedeutend verwendet.

[2] Der Namensbestandteil „Trie“ ist eine Ableitung von „re*trie*val“.

auf Dokumente. Die Kodierung der Referenzen benötigt bei n Dokumenten maximal $\lceil \log_2(n) \rceil$ Bit pro Referenz. Mithilfe von Kompressionsverfahren kann dieser Betrag noch stark verringert werden. Witten u. a. (1999) stellen hierzu eine Reihe von Verfahren vor und vergleichen sie. Gegenüber einer unkomprimierten invertierten Liste, die bis zu 100% der Größe der Dokumentmenge erreichen kann, werden hier Indexgrößen von etwa 4% demonstriert. Der Zeitaufwand für die Kompression und die Dekompression wird allerdings vernachlässigt, obwohl er sich stark auf die Suchzeit auswirkt. Trotman (2003) weist nach, dass stärkere Kompression einen deutlich höheren Dekompressionsaufwand bedeutet. Es muss daher zwischen Kompressionsrate und Dekompressionsgeschwindigkeit abgewogen werden. Den bislang effektivsten Kompromiss zeigen Anh und Moffat (2005). Sie erreichen eine um durchschnittlich 15% schlechtere Kompressionsrate als Witten u. a. (1999), jedoch eine Steigerung der Suchgeschwindigkeit im Index um 40%.

Die Beantwortung einer Wortfrage nach einem Wort w startet im Vokabular. Hier wird der Blattknoten gesucht, der w repräsentiert, und die Referenz auf den korrespondierenden Eintrag in der Liste der Vorkommen gelesen. Anschließend wird der referenzierte Eintrag dekomprimiert und als Ergebnis ausgegeben. Der erste Teil der Suche gelingt in $\mathcal{O}(|w|)$, wobei $|w|$ die Länge des Wortes in Buchstaben ist. Die Laufzeit des zweiten Teils der Suche hängt stark vom eingesetzten Kompressionsverfahren ab.

Die Beantwortung von Kontextfragen erfordert die unabhängige Suche nach allen Einzelworten der Frage und die anschließende Bildung des Durchschnitts aller erhaltenen Ergebnisse. Darüber hinaus muss für jedes der gefundenen Dokumente geprüft werden, ob die Worte nahe genug zusammen stehen bzw. eine Phrase bilden. Um den zusätzlichen Rechenaufwand dafür zu reduzieren, schlagen Williams u. a. (1999) sowie Bahle u. a. (2002) den Nextword-Index vor. Für jedes häufig auftretende Wort w soll eine weitere Liste der Vorkommen für die häufigsten Nachfolger w_N angelegt werden. Sie referenziert alle Dokumente, die ww_N enthalten.

Die erstmalige Konstruktion einer invertierten Liste für eine Dokumentmenge $\mathbf{D}$ erfordert die sequentielle Abarbeitung aller Dokumente. Jedes gelesene Wort w wird im Vokabular nachgeschlagen und anschließend der entsprechende Eintrag in der Liste der Vorkommen aktualisiert. Ist w noch nicht im Vokabular enthalten, wird es aufgenommen und ein neuer Eintrag angelegt.

Einen Vergleich konkreter Algorithmen, die auch die Kompression der invertierten Liste berücksichtigen, zeigen Heinz und Zobel (2003). Nicholas Lester u. a. (2005) beschreiben ein Online-Verfahren, das die Suche auf dem Index zur Konstruktionszeit erlaubt. Dieser Ansatz ist im Vergleich zur Offline-Konstruktion zwar langsamer, aber insbesondere für Indizes in Suchmaschinen interessant.

2.1.2 Suffixbaum

Ein Suffixbaum ist eine Datenstruktur zur Speicherung von Zeichenketten. Eine seiner Schlüsseleigenschaften ist, dass der direkte Zugriff auf alle Suffixe der Zeichenkette ermöglicht wird. Die Baumstruktur indiziert also alle Suffixe im Sinne inverser Indizierung. Der i-te Suffix einer Zeichenkette beginnt beispielsweise beim i-ten Wort und reicht bis zu ihrem Ende. Die Granularität der Suffixe lässt sich steuern, indem anstatt Worten Buchstaben (oder Sätze, Überschriften usw.) betrachtet werden. Die Granularität eines Suffixes bedingt also die Granularität des Indexes.

Der Aufbau eines Suffixbaums wird in Abschnitt 1.1 implizit durch einen einfachen Konstruktionsalgorithmus motiviert. Gusfield (1997) definiert einen Suffixbaum für ein Dokument $\mathbf{d}$ mit $|\mathbf{d}|$ Suffixen als gerichteten Wurzelbaum mit $|\mathbf{d}|$ Blättern. Jeder innere Knoten hat mindestens zwei Kinder und jede Kante ist mit mindestens einem Wort beschriftet. Alle Beschriftungen der ausgehenden Kanten eines Knotens beginnen mit paarweise verschiedenen Worten. Die Verknüpfung der Kantenbeschriftungen auf dem Pfad von der Wurzel zum i-ten Blatt entspricht dem i-ten Suffix in $\mathbf{d}$. Analog ist ein Suffixbaum für eine Dokumentmenge $\mathbf{D}$ definiert. Abbildung 2.2a zeigt einen Suffixbaum für die Beispieldokumente. Im Gegensatz zur invertierten Liste sind alle durch einen Suffixbaum indizierten Zeichenketten vollständig in der Datenstruktur enthalten. Diese Form von Index heißt deshalb selbstindizierend. Die Indizierung einer Dokumentmenge durch einen Suffixbaum bedeutet einen Platzaufwand von mindestens 100% der Dokumentmenge, sofern keine Kompression eingesetzt wird.

Eine Wort- oder Kontextfrage $\mathbf{q}$ kann mit einem Suffixbaum in $\mathcal{O}(|\mathbf{q}| \cdot |w|)$ Zeit beantwortet werden, wobei $|\mathbf{q}|$ die Länge der Frage in Worten und $|w|$ die durchschnittliche Länge eines Wortes ist. Dazu wird der Baum ausgehend vom Wurzelknoten dem Pfad folgend traversiert, dessen Kantenbeschriftungen der Frage entsprechen. Wenn der Baum so weit traversiert wurde, dass die Frage vollständig gefunden wurde, spannt der aktuelle Knoten den Teilbaum des Suffixbaums auf, durch den alle Suffixe repräsentiert werden, die mit dem Fragetext beginnen. Damit sind alle Vorkommen der Frage in der Dokumentmenge gefunden. Wenn kein Pfad existiert, dessen Kantenbeschriftungen der Frage entsprechen, dann ist das gesuchte Wort bzw. sind die gesuchten Worte nicht in der Dokumentmenge enthalten.

Das Suffix-Array von Manber und Myers (1990) wurde parallel zum Suffixbaum entwickelt und erlaubt die Indizierung aller Suffixe eines Dokuments, ohne dass der Text mit in die Datenstruktur aufgenommen wird. Es handelt sich um ein Array, in dem Referenzen auf die Suffixe von Dokumenten in aufsteigender lexikographischer Reihenfolge gespeichert sind. Abbildung 2.2b zeigt ein Suffix-Array. Gegenüber einem Suffixbaum hat es einen deutlichen Platzvorteil, da nur noch Referenzen gespeichert werden. Tatsächlich ist es nach Abouelhoda u. a. (2004) möglich, alle auf Suffixbäumen aufbauenden Verfahren durch Enhanced-Suffix-Arrays ohne Einbußen im Zeitaufwand zu ersetzen. Die Erweiterung besteht aus Arrays derselben Länge,

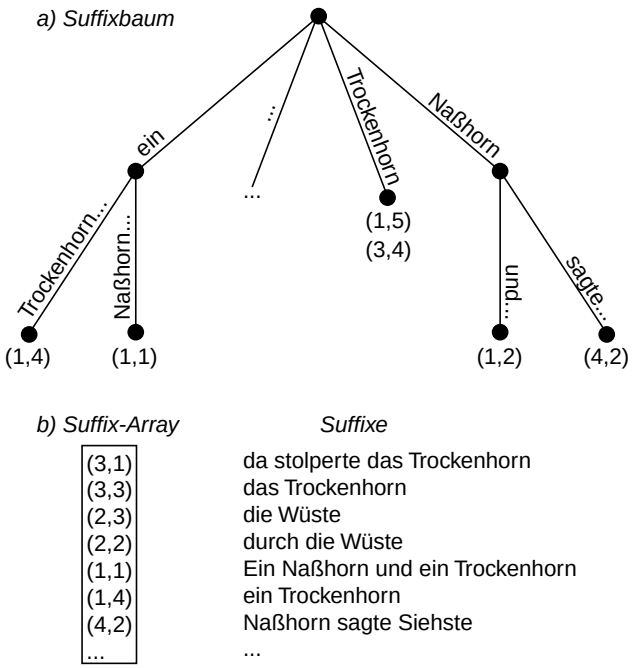

Abbildung 2.2: Suffixbaum (a) und Suffix-Array (b) für die Beispieltexte. Es sind nur die Suffixe im Baum zu sehen, die in mehr als einem Dokument enthalten sind. Die übrigen werden durch die Punkte angedeutet. Die Tupel (i, j) referenzieren das j-te Suffix in Dokument $\mathbf{d}_i$. Im Suffix-Array folgen die Referenzen der lexikographisch aufsteigenden Sortierung der Suffixe.

die verschiedene Zusatzinformationen über die Suffixe und ihre Gemeinsamkeiten kodieren.

Die Anwendung von Kompressionsverfahren auf Suffix-Arrays ermöglicht weitere Platzeinsparungen, so dass nach Grossi u. a. (2004) Indexgrößen von 20% der Größe der Dokumentmenge erreicht werden. Für Suchen auf dem Index müssen hier die Originaldokumente nicht verfügbar sein, da sie im Index mit komprimiert sind. Diese Ergebnisse konkurrieren daher mit denen einer komprimierten invertierten Liste.

Die Konstruktion von Suffixbäumen ist mit dem Konstruktionsalgorithmus von Ukkonen (1995) in $\mathcal{O}(n)$ für n Dokumente möglich. Er basiert auf der inkrementellen Erweiterung (zum Beispiel Wort für Wort) eines Suffixbaums. Der initiale Suffixbaum ist leer, besteht also nur aus dem Wurzelknoten. Für ein Dokument $\mathbf{d} = w_1 \dots w_{|\mathbf{d}|}$ wird in der i-ten Iteration des Algorithmus das Wort w_i dem Suffixbaum hinzugefügt. Alle Suffixe der Wortkette $w_1 \dots w_{i-1}$ sind zu diesem Zeitpunkt bereits im Baum. Das Einfügen von w_i wirkt sich auf das j-te Suffix $w_j \dots w_{i-1}$ wie folgt aus: Von der Wurzel aus wird der Pfad des j-ten Suffixes traversiert. Falls das Suffix nicht an einem Blattknoten endet, wird der Pfad an dieser Stelle geteilt und ein neuer Blattknoten eingefügt. w_i wird der Kantenbeschriftung der Eingangskante des Blattknotens angehängt. Durch „Abkürzungen“ im Baum, die der Algorithmus zur Konstruktionszeit pflegt, wird eine lineare Laufzeit erzielt. In der Praxis werden jedoch häufig weniger komplexe Konstruktionsalgorithmen verwendet. Ein Vergleich von Tian u. a. (2005) belegt, dass auch Algorithmen mit

schlechterer asymptotischer Laufzeit konkurrenzfähig sind.

Suffix-Arrays können ebenfalls in $\mathcal{O}(n)$ konstruiert werden; Kärkkäinen und Sanders (2003), Ko und Aluru (2003) sowie Kim u. a. (2003) haben das unabhängig voneinander nachgewiesen. Der Ko-Aluru-Algorithmus basiert darauf, die Suffixe in disjunkte Mengen zu teilen und dann rekursiv die kleinere der beiden zu sortieren. Dazu werden zwei Typen von Suffixen unterschieden, nämlich diejenigen, die lexikographisch kleiner sind als ihre Nachfolger, und alle übrigen. Anschließend werden die noch unsortierten Suffixe einsortiert. Ein Vergleich konkreter Implementierungen verschiedener Algorithmen ist bei Lee und Park (2004) zu finden. Der Ko-Aluru-Algorithmus schneidet hier am besten ab.

2.1.3 R-Tree

Die Beantwortung von Ähnlichkeitsfragen ist mit keiner der oben genannten Indizierungsverfahren effizient machbar, da die Ähnlichkeitsbeziehungen der indizierten Dokumentmenge nicht wiedergeben werden. Ähnlichkeitsfragen haben zum Ziel, alle Dokumente zu finden, die einem gegebenen Dokument hinreichend ähneln. „Hinreichend“ bedeutet in diesem Zusammenhang, dass die Ähnlichkeit wenigstens $\varphi(\mathbf{d}_i, \mathbf{d}_j) \geq 1-\varepsilon$ beträgt, mit $\varepsilon \in [0, 1]$. Das Vektorraummodell repräsentiert Dokumente als Punkte eines m-dimensionalen Raums. Die Distanz $\sigma_\varphi(\mathbf{d}_i, \mathbf{d}_j)$ des i-ten Dokuments zum j-ten lässt sich aufgrund ihrer Ähnlichkeit $\varphi(\mathbf{d}_i, \mathbf{d}_j)$ errechnen und beträgt $\sigma_\varphi(\mathbf{d}_i, \mathbf{d}_j) = 1 - \varphi(\mathbf{d}_i, \mathbf{d}_j)$. Je ähnlicher sich zwei Dokumente sind, desto näher liegen sie also zusammen. Die Dokumentvektoren aller mit Blick auf ε relevanten Dokumente befinden sich im Vektorraum deshalb in einer ε-Umgebung um das Fragedokument. Ähnlichkeitsfragen können demnach als Fragen nach den Dokumentvektoren eines bestimmten Raumabschnitts aufgefasst werden.

Die Familie der R-Trees, d.h. Varianten des R-Tree nach Guttman (1984), sind räumliche Indizes. Einen detaillierten Überblick über die zahlreichen Varianten geben Manolopoulos u. a. (2003). Im R-Tree werden nahe zusammen liegende Dokumente gruppiert und durch den sie minimal einschließenden Quader repräsentiert. Dasselbe geschieht rekursiv mit allen Quadern, so dass eine Hierarchie entsteht, deren Wurzel ein Quader ist, der alle Dokumente einschließt. Es ist zulässig, dass die Quader einer Ebene sich räumlich überlappen, Dokumente werden jedoch durch genau einen repräsentiert. Die Hierarchie wird mithilfe der Datenstruktur B-Tree gespeichert, so dass jeder Quader einem Knoten im Baum entspricht.

Ein B-Tree (siehe z.B. Cormen u. a. (2001)) ist darauf ausgelegt, Zugriffe auf den sekundären Speicher zu minimieren. Die Größe eines Knotens, und damit sein Ausgangsgrad, ist deshalb im Allgemeinen durch die maximal pro Lesevorgang lesbaren Bytes begrenzt. Im Falle von Festplatten also eine Page. Alle Kinder eines Knotens sind aufgrund eines Ordnungskriteriums sortiert. Der Baum ist zudem balanciert, d.h. alle Blätter sind in derselben Tiefe. Überschrei-

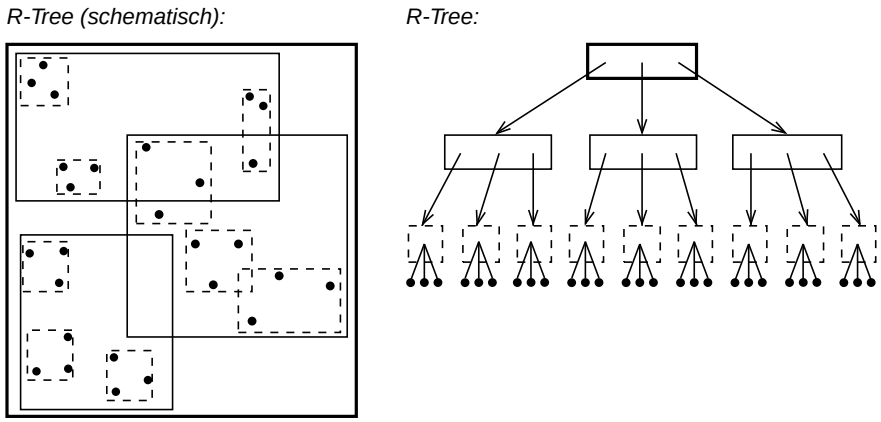

Abbildung 2.3: R-Tree für eine Menge von Punkten im 2-dimensionalen Raum. Die schematische Darstellung zeigt die hierarchische Gruppierung der 2-dimensionalen Quader. Überlappungen sind zu erkennen. Der Baum ist die Repräsentation der schematischen Darstellung als B-Tree. Jeder Knoten entspricht einer Page, jedes Blatt entspricht einem Quader und referenziert alle in ihm enthaltenen Punkte. Der Baum ist balanciert.

tet ein Knoten beim Einfügen eines Objektes in den Baum die Größe einer Page, so wird er geteilt und der Baum neu balanciert. Abbildung 2.3 zeigt schematisch einen R-Tree sowie den korrespondierenden B-Tree.

Das Einfügen eines Dokuments in einen R-Tree geschieht ausgehend von der Wurzel durch rekursive Auswahl eines Kindknotens, dessen Quader am wenigsten vergrößert werden muss, um das Dokument aufzunehmen. Es handelt sich hier um eine Auswahlheuristik, die auf lokaler Optimierung beruht. Die Quader einer Ebene dürfen sich überlappen, weshalb die Auswahl eines lokal schlechter geeigneten Quaders zu einem global besseren Ergebnis führen kann. Das hängt mit der Reihenfolge, in der die Dokumente eingefügt werden, zusammen. Sie bedingt, in welcher Weise die Quader vergrößert werden. Ein optimaler R-Tree verteilt die Dokumente so über den Baum, dass jeder Quader minimal groß ist, also nicht durch Verschiebung eines Dokuments in einen anderen Quader das kumulierte Quadervolumen verringert werden kann. Die eingesetzte Auswahlheuristik sowie die Vermeidung von Überlappungen sind Gegenstand intensiver Forschung und Ursache der zahlreichen Varianten des R-Trees.

Ähnlichkeitsfragen, die wie oben diskutiert auch als Fragen nach Raumabschnitten betrachtet werden können, lassen sich durch rekursives Traversieren des R-Tree beantworten. Dabei werden nur Quader berücksichtigt, die sich mit dem gesuchten Raumabschnitt überschneiden. Bei Erreichen eines Blattknotens werden alle davon referenzierten Dokumente der Ergebnismenge hinzugefügt. Ein Merkmal der Suche im R-Tree ist, dass ihre Precision nicht zwangsläufig 1 ist, da auch Dokumente gefunden werden, die nahe dem gesuchten Raumabschnitt liegen, aber nicht in ihm enthalten sind. Sie müssen nachträglich durch einen sequentiellen Vergleich mit dem Fragedokument herausgefiltert werden.

Die Dimensionalität m des Raums, auf dem räumliche Suchen stattfinden, hat großen Einfluss

auf die Suchgeschwindigkeit im R-Tree. Weber u. a. (1998) haben verschiedene räumliche Indizes diesbezüglich untersucht, darunter auch Varianten des R-Trees. Sie kommen zu dem Ergebnis, dass ab einer Dimensionalität $m \approx 10$ alle Indizes im Durchschnitt langsamer durchsuchbar sind, als mit einer sequentiellen Suche. Dieses Verhalten steht in Beziehung zum so genannten „Fluch der Dimensionalität“. Dokumente, die durch das Vektorraummodell repräsentiert werden, haben oftmals eine um Größenordnungen höhere Dimensionalität als 10. Das disqualifiziert die Varianten des R-Trees für den Einsatz als Indexstruktur für Dokumente.

2.2 Transformative Indizierung

Die Vertreter transformativer Indizierung sind die Signaturliste und das Similarity-Hashing. Beide verwenden als Transformationsvorschrift Hashfunktionen. Im folgenden Abschnitt werden die Grundlagen des Hashings daher zunächst aufgearbeitet, gefolgt von einer detaillieren Diskussion der beiden Indizierungsverfahren.

2.2.1 Hashtabellen und Hashfunktionen

Hashtabellen basieren auf Arrays. Ein Array ist eine Datenstruktur, in der eine begrenzte Anzahl aufeinander folgender Speicherplätze fester Größe aufsteigend durchnummeriert sind. Jeder Speicherplatz ist dabei groß genug ausgelegt, um einen Wert einfacher Datentypen oder Referenzen auf Objekte aufzunehmen. Der direkte Zugriff darauf ist durch die jeweilige Indexnummer des Speicherplatzes gewährleistet und benötigt konstante Zeit. Jedem Eintrag im Array ist also genau eine Indexnummer zugeordnet, die als Schlüssel für den Zugriff dient und bei gezielten Zugriffen daher a-priori bekannt sein muss.

Wenn die Anzahl möglicher Schlüssel sehr groß ist, aber nur eine kleine Teilmenge von ihnen tatsächlich benötigt wird, sind Arrays nicht optimal im Speicherverbrauch. Da im ungünstigsten Fall nicht bekannt ist, welche Schlüssel zur Laufzeit tatsächlich belegt werden, muss ein Array zu jeder Zeit groß genug ausgelegt werden, also Speicherplätze für alle möglichen Schlüssel reservieren. Das ist offensichtlich ineffizient und kann mithilfe von Hashtabellen umgangen werden, ohne dass sich der Aufwand für Zugriffe stark erhöht.

Eine Hashtabelle T besteht aus einem Array mit $|T|$ Speicherplätzen und einer Hashfunktion[3] $h : \mathbb{N} \to \{0 \dots |T|\}$, wobei die natürlichen Zahlen $\mathbb{N}$ als potenzielle Schlüssel betrachtet werden. Die Hashfunktion erlaubt die Benutzung aller Schlüssel, unabhängig von der Größe des Arrays. Jeder Zugriff mit einem Schlüssel i führt zuerst zur Berechnung seines Hashwerts $h(i)$, der den Speicherplatz im Array angibt. Ein Speicherplatz heißt in diesem Zusammenhang auch Bucket. Es ist leicht zu erkennen, dass h nicht injektiv ist, da weniger Buckets existieren, als

Schlüssel vorhanden sind. Das Ereignis eines anderen Schlüssels j mit demselben Hashwert wie dem von Schlüssel i, also formal $h(i) = h(j)$, heißt Hashkollision. Im Falle einer Kollision gibt es verschiedene Möglichkeiten, sie robust zu behandeln. Die einfachste ist die Verkettung aller Daten, deren Schlüssel auf denselben Bucket abgebildet werden. Die verkettete Liste wird dann anstelle der Daten in der Hashtabelle abgelegt. Wenn bekannt ist, dass maximal $|T|$ Daten gespeichert werden, kann im Falle einer Kollision auf alternative Hashfunktionen ausgewichen werden, bis ein freier Bucket gefunden wurde. Dieses Vorgehen heißt offene Adressierung.

Eine gute Hashfunktion hat nach Cormen u. a. (2001) die Eigenschaft, dass die Wahrscheinlichkeit, mit der ein beliebiger Schlüssel einem bestimmten Bucket zugeordnet wird, unabhängig von allen anderen Schlüsseln, $1/|T|$ beträgt. Die Konstruktion einer solchen Hashfunktionen kann zum Beispiel heuristisch unter Einbeziehung von Domäneninformationen geschehen. Das ermöglicht häufig eine nahezu optimale Verteilung aller auftretenden Schlüssel. Ist über die Domäne nichts bekannt, kann die Divisions- oder die Multiplikationsmethode angewandt werden. Erstere definiert eine Hashfunktion $h(i) = i \bmod |T|$ für einen Schlüssel i und eine Hashtabelle mit $|T|$ Buckets. Geeignete Werte für $|T|$ sind oftmals Primzahlen, da mit ihnen alle Bits von i in die Division einbezogen werden. Wird $|T|$ beispielsweise als gerade Zahl oder Potenz von 2 gewählt, entstehen weitreichende Abhängigkeiten zwischen Schlüsseln und ihren Hashwerten. Das kann zu einer größeren Häufigkeit von Kollisionen führen.

Mit der Multiplikationsmethode werden Hashfunktionen nach folgendem dem Muster definiert:

$$h(i) = \lfloor |T|(i \cdot c - \lfloor i \cdot c \rfloor) \rfloor \qquad \text{mit } c \in (0, 1)$$

Der gebrochene Anteil von $i \cdot c$ wird mit $|T|$ multipliziert. Die Wahl des Parameters $|T|$ ist hier gegenüber der Divisionsmethode unkritisch. Knuth (1998) zufolge ergibt sich mit $c \approx 0.62$ häufig eine gleichmäßige Zuordnung der Schlüssel auf Buckets.

Die Häufigkeit von Kollisionen wird bedingt durch die verwendete Hashfunktion und die Verteilung auftretender Schlüssel. Für alle Hashfunktionen existiert per Definition je eine Menge von Schlüsseln, deren Hashwerte kollidieren. Daraus folgt im Worst-Case für eine Hashtabelle mit fester Hashfunktion eine Zugriffszeit von $\mathcal{O}(n)$, bei n verschiedenen Schlüsseln. Universal-Hashing nach Carter und Wegman (1977) entschärft dieses Problem mittels Randomisierung. Die Definition einer Hashtabelle wird hier erweitert, so dass anstatt einer einzelnen Hashfunktion eine universale Menge oder auch Familie $\mathcal{H}$ unterschiedlicher Hashfunktionen eingesetzt wird. Zur Initialisierungszeit wird zufällig eine Hashfunktionen ausgewählt. Der Worst-Case

[3] Die Definition von Hashfunktionen ist zur Verständnisförderung auf die Schlüsselmenge $\mathbb{N}$ beschränkt. Diese Einschränkung ist jedoch nicht bindend, sondern dient als generischer Startpunkt aller Erwägungen. Es wird in der Praxis davon ausgegangen, dass für alle Datentypen und Objekte, also auch für Dokumente bzw. ihre Modelle, eine sinnvolle Abbildung auf $\mathbb{N}$ aufgestellt werden kann, so dass sie mit den oben definierten Hashfunktionen verarbeitet werden können.

wird zwar dadurch nicht verhindert, es ist allerdings unwahrscheinlich, dass er für eine Menge von Schlüsseln wiederholt eintritt.

Eine Familie $\mathcal{H}$ von Hashfunktionen wird universal genannt, wenn für alle Schlüsselpaarungen i, j mit $i \neq j$ höchstens $|\mathcal{H}|/|T|$ Hashfunktionen in $\mathcal{H}$ existieren, für die die Hashwerte beider kollidieren. Für den Schlüssel i allein lässt sich damit bei insgesamt n zu hashenden Schlüsseln und einer zufällig gewählten Hashfunktion eine durchschnittliche Kollisionshäufigkeit von höchstens $n/|T|$ nachweisen. Die Zugriffsgeschwindigkeit beträgt also im Durchschnitt $\mathcal{O}(1 + n/|T|)$, was einer optimalen Verteilung der Schlüssel auf die vorhandenen Buckets entspricht.

2.2.2 Signaturliste

Eine Signaturliste basiert auf einer Hashfunktion $h : W \rightarrow \{0, 1\}^k$, die Worte auf Bitfolgen abbildet. Diese Bitfolgen dienen als Signaturen oder auch Fingerabdrücke der Worte. Die Signatur eines Dokuments $\mathbf{d}$ wird durch Überlagerung der Signaturen aller Worte in $\mathbf{d}$ erzeugt. Das geschieht durch Anwendung einer bitweisen ODER-Verknüpfung, so dass das i-te Bit genau dann gesetzt wird, wenn das i-te Bit in mindestens einer Wortsignatur gesetzt ist. Ein Dokument wird auf diese Weise ebenfalls auf eine k-stellige Bitfolge abgebildet. Die Signaturen einer Menge $\mathbf{D}$ von Dokumenten werden in einer Datei gespeichert, in der die i-te Signatur dem i-ten Dokument aus $\mathbf{D}$ entspricht.

Zur Beantwortung einer Wortfrage nach w wird die Bitfolge $h(w)$ berechnet und mit allen Signaturen von $\mathbf{D}$ verglichen. Wenn die bitweise UND-Verknüpfung von $h(w)$ mit der Signatur eines Dokuments $\mathbf{d}$ gleich $h(w)$ ist, wird angenommen, dass w in $\mathbf{d}$ enthalten ist. Wenn w in $\mathbf{d}$ enthalten ist, dann ist $h(w)$ per Definition Teil der überlagerten Signatur von $\mathbf{d}$. Wenn w nicht in $\mathbf{d}$ enthalten ist, kann jedoch ein Wort w' enthalten sein, für das $h(w') = h(w)$ gilt, oder eine Teilmenge von Worten, deren Überlagerung die Signatur $h(w)$ enthält. Solche falsch positiv bewerteten Dokumente werden auch false-drops genannt, die nachträglich identifiziert werden müssen. Die Beantwortung von Kontextfragen läuft analog: Die Signatur der Frage ist die Überlagerung der Worte, nach denen gefragt wird. Enthält ein Dokument alle nachgefragten Worte, enthält seine Signatur die der Kontextfrage. Im Anschluss muss noch überprüft werden, ob die Worte in der gewünschten Reihenfolge im Dokument vorkommen. Abbildung 2.4 zeigt eine Signaturliste und den Ablauf einer Suche.

Der Recall einer Suche in einer Signaturliste beträgt für Wort- und Kontextfragen immer 1. Die Precision schwankt mit der Wahrscheinlichkeit P für das Auftreten von false-drops. P lässt sich wie folgt ermitteln: Angenommen, h setzt für jedes Wort w in $h(w)$ zufällig t Bit. Für eine k Bit lange Signatur ergibt sich ein Anteil gesetzter Bits von $\tau = t/k$. Die Wahrscheinlichkeit, dass ein bestimmtes Bit in einer Signatur für ein Dokument $\mathbf{d}$ der Länge $|\mathbf{d}|$ zufällig gesetzt ist, lässt

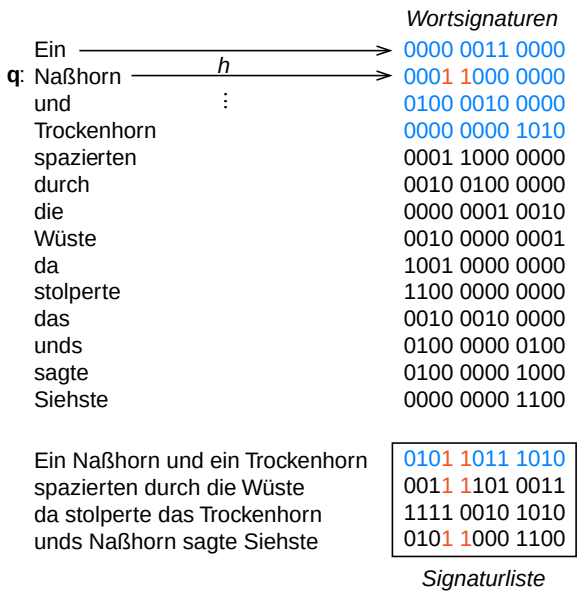

Abbildung 2.4: Die Signaturliste für die Beispieldokumente enthält die überlagerten Wortsignaturen der Worte jedes Dokuments. Die Signatur des ersten Dokuments besteht beispielsweise aus den ersten vier Wortsignaturen (blau markiert). Die Frage $\mathbf{q}$ nach dem Wort „Naßhorn“ ergibt, dass Dokument 1,2 und 4 das Wort enthalten (rot markiert). Tatsächlich kollidiert aber im zweiten Dokument die Signatur von „spazierten“ mit der von „Naßhorn“. Es handelt sich also um ein falsch positives Ergebnis. Der Fall, dass eine Überlagerung von Wortsignaturen zufällig die Signatur eines dritten Wortes enthält, tritt bei „und“ sowie „Trockenhorn“ für „sagte“ ein.

sich über die Gegenwahrscheinlichkeit $1 - \frac{1}{k}$ errechnen, dieses Bit bei einer zufälligen Auswahl nicht zu wählen:

$$1 - \left(1 - \frac{1}{k}\right)^{|\mathbf{d}| \cdot t} \approx 1 - e^{-|\mathbf{d}| \cdot \tau}$$

Daraus folgt:

$$P = (1 - e^{-|\mathbf{d}| \cdot \tau})^t$$

Wenn beispielsweise $\tau = \ln(2)/|\mathbf{d}|$ gewählt wird, ergibt sich für $t = k \cdot \ln(2)/|\mathbf{d}|$ ein konkretes $P = 1/2^t$. Die Vergrößerung der Signaturlänge k senkt daher P, steigert aber den Platzaufwand der Signaturliste für n Dokumente in der Größenordnung $\mathcal{O}(n \cdot k)$. Auf diese Weise kann k abhängig von einer gewünschten Wahrscheinlichkeit P für das Auftreten eines false-drops gewählt werden.

Bei Verwendung der Negation in booleschen Fragen tritt der Sonderfall ein, dass die Precision 1 ist, der Recall aber mit der Wahrscheinlichkeit für falsch negativ deklarierte Dokumente schwankt. Bei den nicht gefundenen Dokumenten handelt es sich um genau diejenigen, die bei der Suche ohne Negation falsch positiv deklariert würden.

Einen ausführlichen Überblick über verschiedene Varianten der Signaturliste geben Faloutsos (1992) und Lee u. a. (1995). Die bekannteste unter ihnen ist die in Bitscheiben zerteilte Signa-

turliste. Sie reduziert den Suchaufwand nach der Signatur $h(\mathbf{q})$ einer Frage $\mathbf{q}$ von $\mathcal{O}(n \cdot k)$ auf $\mathcal{O}(t \cdot n)$ bei $t \ll k$ gesetzten Bits in $h(\mathbf{q})$. Die Partitionierung geschieht durch Aufteilung der n Signaturen in k einzelne Dateien, so dass der i-te Eintrag der j-ten Datei dem j-ten Bit der i-ten Signatur entspricht. Die Suche nach den Signaturen, die zu $h(\mathbf{q})$ passen, lässt sich so auf die t gesetzten Bits beschränken. Für zwei Bits i_1 und i_2 verläuft die Suche wie folgt: Zunächst werden alle Positionen in Datei i_1 ermittelt, die ebenfalls gesetzt sind. Aus dieser Teilmenge von Dokumentsignaturen werden anschließend alle Positionen entfernt, für die in Datei i_2 kein Bit gesetzt ist. Die Rekursion wird so lange fortgeführt, bis die Dateien aller t gesetzten Bits aus $h(\mathbf{q})$ überprüft wurden. Die dann verbleibenden Positionen bilden die Antwortmenge.

Über die in Signaturlisten eingesetzten Hashfunktionen hinaus spielen andere Kompressionsverfahren keine Rolle. Die Wortsignaturen selbst sind bereits eine Form verlustbehafteter Kompression, die so stark ist, dass die Berechnung des Ursprungswortes aus einer Signatur nicht mehr möglich ist. Mit der Überlagerung aller Wortsignaturen eines Dokuments ist wiederum eine verlustbehaftete Kompression verbunden. Eine weitere Kompression von Signaturen wäre nur dann sinnvoll, wenn die Signaturlänge deutlich vom Optimum abweicht, sie also größer als notwendig ist. Dann nämlich enthalten Signaturen deutlich mehr Nullen als Einsen. Werden die Signaturen jedoch mithilfe der oben vorgestellten Formeln gebildet, ist keine weitere verlustlose Kompression möglich, da der Informationsgehalt einer Signatur mit nahezu gleichmäßig vielen Nullen und Einsen bereits optimal ist.

Die Signaturliste steht seit ihrer Erfindung in Konkurrenz zur invertierten Liste. Dennoch wurde der erste umfassende Vergleich beider erst von Zobel u. a. (1998) durchgeführt. Sie schließen mit dem Ergebnis, dass die invertierte Liste der Signaturliste in nahezu allen Belangen überlegen ist. Das ist insbesondere darauf zurückzuführen, dass aktuelle Varianten der invertierten Liste weniger Platz beanspruchen als Signaturlisten. Letztere lassen sich nur langsamer konstruieren, aktualisieren und durchsuchen. Darüber hinaus besteht insbesondere bei in Bitscheiben zerteilten Signaturlisten das Problem schlechter Datenlokalität. Das Durchsuchen von t Dateien erfordert $\mathcal{O}(t)$ Festplattenzugriffe, wohingegen bei der invertierten Liste einzig das Vokabular durchsucht wird. Auf theoretischer Seite haben Signaturlisten den Nachteil, dass die durchschnittliche Größe eines Dokuments $|\mathbf{d}|$ im Vorhinein bekannt sein muss, um die Signaturlänge optimal einstellen zu können. Weitere Studien von Garratt u. a. (2001) und Helmer und Moerkotte (2003) bestätigen die obigen Ergebnisse.

Zur Verbesserung der Suchgeschwindigkeit gegenüber der sequentiellen Suche in der Liste aller Dokumentsignaturen schlägt Chen (2005) den verallgemeinerten Signaturbaum vor. Er basiert auf dem Signaturbaum nach Chen (2002), einem binären Baum, der analog zum Vokabular einer invertierten Liste, als Index für die Einträge der Signaturliste dient. Jeder innere Knoten speichert eine Zahl $j \in \{1, \ldots, k\}$, mit der das j-te Bit aller Signaturen gemeint ist. Die ausgehenden Kanten innerer Knoten tragen entweder die Beschriftung 0 (linkes Kind) oder 1 (rechtes Kind), entsprechend den möglichen Werten eines Bits. Blattknoten referenzieren Signaturen.

Zum Einfügen der i-ten Signatur in den Signaturbaum wird der Baum von der Wurzel aus traversiert, so dass an jedem Knoten das j-te Bit der Signatur gelesen und entsprechend seinem Wert die linke bzw. rechte Kante verfolgt wird. Ein Knoten v wird erreicht, der entweder ein innerer Knoten ist und keine passende Ausgangskante hat, oder ein Blattknoten. Im ersten Fall wird ein Blattknoten erzeugt, der die i-te Signatur referenziert, und v angehängt. Im zweiten Fall, wird ein Bit j gesucht, in dem sich die i-te und die von v referenzierte Signatur unterscheiden. Ein neuer Knoten wird erzeugt, der j speichert und v als Kindknoten bekommt. Danach wird wie im ersten Fall verfahren.

Im verallgemeinerten Signaturbaum wird dieses Prinzip verallgemeinert. In jedem inneren Knoten werden hier anstatt einem bis zu $|v|$ Bit adressiert. Die Beschriftungen der Ausgangskanten sind damit bis zu $|v|$ Bit lang, so dass jeder Knoten bis zu $2^{|v|}$ Kindknoten hat. So kann zwischen Vergleichsaufwand je Knoten und der Tiefe des Baums abgewogen werden, was insbesondere Einfluss auf die Häufigkeit des Zugriffs auf den sekundären Speicher hat.

Um alle Signaturen zu finden, die auf eine Fragesignatur $h(\mathbf{q})$ passen, wird ein Signaturbaum von der Wurzel ausgehend wie folgt traversiert: Wenn am Knoten v das j-te Bit in $h(\mathbf{q})$ gleich 1 ist, wird die Suche nur beim rechten Kindknoten fortgesetzt, andernfalls bei beiden. Im ersten Fall werden alle Signaturen ausgeschlossen, die eine 0 dort haben, wo $h(\mathbf{q})$ eine 1 hat. Andernfalls müssen alle Kindknoten traversiert werden, da eine 0 in $h(\mathbf{q})$ keine Aussage darüber zulässt, ob die Signaturen im von v aufgespannten Teilbaum $h(\mathbf{q})$ enthalten sind oder nicht.

2.2.3 Similarity-Hashing

Wir definieren den Begriff „Similarity-Hashing“ als Oberbegriff für die von Indyk und Motwani (1998) vorgeschlagene Idee, Hashkollisionen als Indikator für Ähnlichkeit zu interpretieren. Das Ereignis einer Hashkollision zweier Dokumente tritt mit Hashfunktionen, wie sie in Abschnitt 2.2.1 definiert sind, im Optimalfall rein zufällig auf. Diese Idee ist mit solchen Hashfunktionen deshalb nicht vereinbar, da Dokumente mit ihnen möglichst gleichmäßig über eine Hashtabelle verteilt werden, unabhängig von den Beziehungen der Dokumente untereinander. Das Ereignis einer Hashkollision zweier Dokumente erlaubt daher keinen Rückschluss auf ihre Ähnlichkeit im Sinne transformativer Indizierung.

Die Anforderungen an Hashfunktionen im Similarity-Hashing wurden deshalb modifiziert. Hier wird gefordert, dass die Verteilung der Dokumente auf Buckets von ihren Ähnlichkeitsbeziehungen untereinander abhängig ist. Ein Dokument soll nur dann einem bestimmten Bucket zugeordnet werden, wenn diesem auch ähnliche Dokumente zugeordnet werden. Mit anderen Worten soll eine Hashfunktion h die auf der Dokumentmenge $\mathbf{D}$ definierte Ähnlichkeitsfunktion φ widerspiegeln. Das bedeutet, dass die Wahrscheinlichkeit $P[h(\mathbf{d}_1) = h(\mathbf{d}_2)]$, mit der die Hashwerte zweier Dokumente $\mathbf{d}_1$ und $\mathbf{d}_2$ kollidieren, mit ihrer Ähnlichkeit $\varphi(\mathbf{d}_1, \mathbf{d}_2)$ wächst.

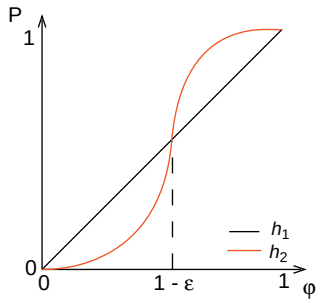

Abbildung 2.5: Der Zusammenhang zwischen Kollisionswahrscheinlichkeit P der Hashwerte von Dokumenten zu ihrer Ähnlichkeit φ für zwei Hashfunktionen h_1 und h_2. Für einen vordefinierten Ähnlichkeitsschwellwert ε ist h_2 der Funktion h_1 überlegen, da die Hashwerte von Dokumenten mit großer Ähnlichkeit bei dieser Funktion mit größerer Wahrscheinlichkeit kollidieren und umgekehrt.

Eine Hashtabelle T, die h verwendet, gruppiert auf diese Weise ein Dokument mit größerer Wahrscheinlichkeit mit ähnlichen Dokumenten.

T lässt sich damit als Index für eine Dokumentmenge $\mathbf{D}$ verwenden, wenn jedes Dokument $\mathbf{d} \in \mathbf{D}$ anhand $h(\mathbf{d})$ einem Bucket zugeordnet wird. Die Beantwortung einer Ähnlichkeitsfrage nach einem Dokument $\mathbf{q}$ geschieht durch das Auslesen aller Dokumente im Bucket $T[h(\mathbf{q})]$. Diese Operation erfordert die einmalige Berechnung des Hashwertes von $\mathbf{q}$, und ist damit unabhängig von der Anzahl n indizierter Dokumente. Die im ausgelesenen Bucket enthaltenen Dokumente ähneln per Definition der Hashfunktion mit großer Wahrscheinlichkeit $\mathbf{q}$.

Damit mit diesem Indizierungsverfahren, analog zum R-Tree, Fragen nach Dokumenten eines bestimmten Raumabschnittes möglich sind, ist die Steuerung der Hashfunktionen abhängig von einem Schwellwert ε nötig. Gefunden werden sollen dann alle Dokumente in einer ε-Umgebung um das Fragedokument. Übertragen auf das Konzept der Ähnlichkeit, gibt $\varepsilon \in [0, 1]$ an, wie stark die Ähnlichkeit $\varphi(\mathbf{d}_1, \mathbf{d}_2)$ zweier Dokumente $\mathbf{d}_1$ und $\mathbf{d}_2$ maximal von 1 abweichen darf, damit sie ein und demselben Bucket zuzuordnen sind. Abbildung 2.5 illustriert, wie sich die Wahl von ε auf die Kollisionswahrscheinlichkeit zweier unterschiedlicher Hashfunktionen auswirken könnte. Die Wahrscheinlichkeit sollte im Allgemeinen groß sein, wenn zwei Dokumente eine Ähnlichkeit größer als $1 - \varepsilon$ haben und andernfalls gering. Eine optimale Hashfunktion würde für Ähnlichkeiten kleiner $1 - \varepsilon$ die Wahrscheinlichkeit 0 und andernfalls die Wahrscheinlichkeit 1 aufweisen. In der Praxis jedoch wird ε häufig nicht direkt, sondern nur implizit über Parameter der Hashfunktionen eingestellt.

Die Wahl von ε dient gleichzeitig der Bewertung der Retrieval-Eigenschaften einer Hashfunktion h, also der Messung von Precision und Recall. Als relevant sind all jene Dokumente zu betrachten, die zu einem Fragedokument wenigstens eine Ähnlichkeit von $1 - \varepsilon$ aufweisen. Es ist offensichtlich, dass die Precision im Similarity-Hashing nicht immer 1 sein kann, denn es gibt hier sowohl eine Wahrscheinlichkeit für das Auftreten, als auch für das Unterbleiben einer Hashkollision der Hashwerte relevanter Dokumente. Das Gleiche gilt für den Recall.

Eine gängige Methode zur Steigerung des Recalls besteht darin, das Verfahren wiederholt anzuwenden. Das heißt, dass die Dokumente $\mathbf{D}$ durch l Hashtabellen mit verschiedenen Hashfunktionen indiziert werden. Die Antwort auf ein Fragedokument $\mathbf{q}$ besteht dann aus der Vereinigung aller l Buckets aus den Hashtabellen, zu denen $\mathbf{q}$ Schlüssel ist. Es ist zu erkennen, dass die Anzahl durchschnittlich gefundener Dokumente mit der Anzahl der eingesetzten Hashfunktionen steigt. Tatsächlich stellen wir hier einen monotonen Zusammenhang fest: Es werden nicht weniger Dokumente gefunden, wenn die Anzahl eingesetzter Hashfunktionen steigt. Die Beeinflussung der Precision geschieht dagegen durch die Vergrößerung oder Verkleinerung der Genauigkeit der Hashwerte. Je genauer sie sind, desto größer wird auch die Precision, da die Dokumente besser durch sie repräsentiert werden Analog gilt auch hier eine Monotonie: Es werden nicht mehr Dokumente gefunden, wenn die Genauigkeit der Hashwerte gesteigert wird. Gleichzeitig verursacht die Steigerung der Genauigkeit jedoch eine Verringerung des Recalls und analog verursacht der Einsatz von mehr Hashfunktionen eine Verringerung der Precision.

Similarity-Hashing ist das einzige der vorgestellten Indizierierungsverfahren, bei dem sowohl die Precision als auch der Recall einer Suchanfrage auf einer indizierten Dokumentkollektion von 1 abweichen kann. Das ermöglicht einerseits die Steigerung der Suchgeschwindigkeit, nämlich die einmalige Berechnung des Hashwertes für $\mathbf{q}$, andererseits ist das Suchergebnis unter Umständen nicht vollständig bzw. enthält irrelevante Ergebnisdokumente. Letztere werden dann, analog zum R-Tree und der Signaturliste, durch sequentiellen Vergleich mit $\mathbf{q}$ identifiziert. Dieser Aufwand dominiert die Suchgeschwindigkeit in der Größenordnung der durchschnittlichen Größe eines Buckets. In der Praxis ergibt sich daraus ein linearer Aufwand für die Suche in n Dokumenten von $\mathcal{O}(c \cdot n)$ mit $c \ll 1$. c ist der Anteil durchschnittlich gefundener Dokumente an der indizierten Kollektion. Die asymptotische Laufzeit einer Suche nach Dokumenten ist hier also nicht besser als die einer sequentiellen Suche. Sie ist jedoch unabhängig von der Dimensionalität des Vektorraummodells, durch das die Dokumente repräsentiert werden, und damit ermöglicht der einstellbare Faktor c die Reduzierung der tatsächlich zu vergleichenden Dokumente je Suche.

In der Folge trägt im Similarity-Hashing hauptsächlich die vorausschauende Definition und Konfiguration der eingesetzten Hashfunktionen zum Erfolg bei. Im folgenden Kapitel werden die zwei bis dato bekannten Ansätze hierfür detailliert vorgestellt. Dabei handelt es sich um Locality-Sensitive-Hashing von Indyk und Motwani (1998) und Fuzzy-Fingerprinting von Stein (2005a).

2.3 Zusammenfassung

Zum Vergleich der vorgestellten Indizierungsverfahren miteinander zeigt Abbildung 2.6 eine Gegenüberstellung von Retrieval-Eigenschaften und algorithmischen Kriterien, aufgeschlüsselt

Fragetyp / Indizierungsverfahren	Algorithmische Kriterien: Suchzeit	Algorithmische Kriterien: Platzverbrauch	Retrieval-Eigenschaften: Precision	Retrieval-Eigenschaften: Recall
Wortfragen				
Invertierte Liste	$\mathcal{O}(\|w\|)$	$\mathcal{O}(b \cdot n)$	1	1
Suffixbaum	$\mathcal{O}(\|w\|)$	$\mathcal{O}(\|w\| \cdot \|\mathbf{d}\| \cdot n)$	1	1
Signaturliste	$\mathcal{O}(\tau \cdot n)$	$\mathcal{O}(k \cdot n)$	$[0, 1]$	1
Kontextfragen				
Invertierte Liste	$\mathcal{O}(\|\mathbf{q}\| \cdot \|w\|)$	$\mathcal{O}(b \cdot n)$	1	1
Suffixbaum	$\mathcal{O}(\|\mathbf{q}\| \cdot \|w\|)$	$\mathcal{O}(\|w\| \cdot \|\mathbf{d}\| \cdot n)$	1	1
Signaturliste	$\mathcal{O}(\tau \cdot n)$	$\mathcal{O}(k \cdot n)$	$[0, 1]$	1
Ähnlichkeitsfragen ($m \leq 10$)				
R-Tree	$\mathcal{O}(\log n)$	$\mathcal{O}(\|v\| \cdot \|V\|)$	$[0, 1]$	1
Similarity-Hashing	$\mathcal{O}(c \cdot n)$, $c \ll 1$	$\mathcal{O}(k \cdot l \cdot n)$	$[0, 1]$	$[0, 1]$
Ähnlichkeitsfragen ($m > 10$)				
Sequentielle Suche	$\mathcal{O}(n)$	—	1	1
Similarity-Hashing	$\mathcal{O}(c \cdot n)$, $c \ll 1$	$\mathcal{O}(k \cdot l \cdot n)$	$[0, 1]$	$[0, 1]$

Legende

n	Anzahl indizierter Dokumente	$\|\mathbf{q}\|$	Fragelänge in Worten
$\|\mathbf{d}\|$	Durchschn. Dokumentlänge in Worten	l	Anzahl der Hashfunktionen
$\|w\|$	Durchschn. Wortlänge in Buchstaben	b	Bits pro Dokumentreferenz
k	Bits pro Hashwert	$\|V\|$	Knotenanzahl im R-Tree abh. von n
τ	Anteil gesetzter Bits an k	$\|v\|$	Bits pro Knoten
c	Durchschn. Anzahl gefundener Dokumente	m	Dimensionalität der Dokumentmodelle

Abbildung 2.6: Gegenüberstellung von Retrieval-Eigenschaften und algorithmischen Bewertungskriterien der vorgestellten Indizierungsverfahren abhängig von den zu beantwortenden Fragetypen. Für die invertierte Liste wurde eine Indizierung auf Dokumentebene angenommen, so dass die Wortpositionen im Dokument nicht indiziert werden. Das erlaubt den direkten Vergleich mit der Signaturliste, bei der es genauso ist. Für die Signaturliste wurde eine in Bitscheiben getrennte Speicherung angenommen.

nach Fragetypen. Zu erkennen ist, dass die invertierte Liste bei Wortfragen eine überlegene Suchzeit und einen überlegenen Platzverbrauch aufweist. Die Anzahl der Bits pro Dokumentreferenz b ist in der Praxis deutlich kleiner als die Länge k des Hashwertes für ein Dokument. Zudem kann mit einer Signaturliste kein Recall von 1 garantiert werden. Der Platzverbrauch eines Suffixbaums wurde hier konservativ geschätzt. Die alternative Repräsentationsform Suffix-Array verspricht hier deutliche Verbesserungen, insbesondere mit Hinblick auf den Einsatz von Kompressionsalgorithmen.

Im Falle von Kontextfragen verschlechtert sich die Suchzeit in der invertierten Liste, die der Signaturliste bleibt dagegen gleich, da hier die gleichzeitige Suche nach mehreren Begriffen im Dokument möglich ist. Der Suffixbaum hat bei Kontextfragen nur im Falle von Phrasen Vorteile, da eine Phrase der Beginn eines Suffixes in den Dokumenten ist, die sie enthalten.

Bei Ähnlichkeitsfragen zeigt sich für Daten mit einer geringen Dimensionalität $m \leq 10$, dass der R-Tree dem Similarity-Hashing sowohl algorithmisch als auch von den Retrieval-Eigen-

schaften her überlegen ist. Steigt die Dimensionalität jedoch, ist die Situation umgekehrt. Der R-Tree arbeitet hier langsamer als der Worst-Case einer sequentiellen Suche, so dass diese als Referenz dient. Demgegenüber ermöglicht Similarity-Hashing eine deutliche Verringerung der Suchzeit.

Kapitel 3

Hashfunktionen im Similarity-Hashing

Die in diesem Kapitel vorgestellten Verfahren zum Similarity-Hashing sind Locality-Sensitive-Hashing (LSH) und Fuzzy-Fingerprinting (FF). LSH ist ein Rahmenwerk für Similarity-Hashing, das auf Randomisierung beruht. Es ist für hochdimensionale Vektorräume konzipiert und dient der Lösung von Nächster-Nachbar-Problemen. Deshalb ist die Übertragung des Konzeptes Ähnlichkeit auf das der Distanz notwendig. Die Konzepte sind diametral zueinander, so dass hohe Ähnlichkeit niedrige Distanz bedeutet und umgekehrt. Jeder Ähnlichkeitsbetrag lässt sich auch als Distanz auffassen, indem er von 1 subtrahiert wird. Die Verwendung des Vektorraummodells zur Repräsentation von Dokumenten ist hier Voraussetzung.

Die Retrieval-Eigenschaften von LSH sind bis zu einem gewissen Grad beweisbar, denn es ist möglich, eine untere Schranke für die Wahrscheinlichkeit anzugeben, dass der Hashwert eines Punktes mit dem eines anderen kollidiert, wenn sich dieser innerhalb seiner ε-Umgebung befindet. In den folgenden Abschnitten wird das Rahmenwerk und zwei der darin verwendeten Familien von Hashfunktionen vorgestellt.

Fuzzy-Fingerprinting beschreibt eine Familie von Hashfunktionen, die Domänenwissen des textbasierten Information-Retrieval in die Konstruktion von Hashwerten integriert. Es werden also Informationen über die Art der zu indizierenden Daten ausgenutzt. Dadurch soll eine Verbesserung der Retrieval-Eigenschaften gegenüber dem uninformierten LSH erreicht werden.

3.1 Konstruktionsprinzipien von Hashfunktionen

Zur Konstruktion einer Hashfunktion für das Similarity-Hashing müssen die zu hashenden Objekte (hier: Dokumente) auf die natürlichen Zahlen abgebildet werden. Die Analyse der Problemstellung und der bisher bekannten Hashfunktionen haben gezeigt, dass dazu Antworten auf die folgenden vier Fragestellungen zu finden sind:

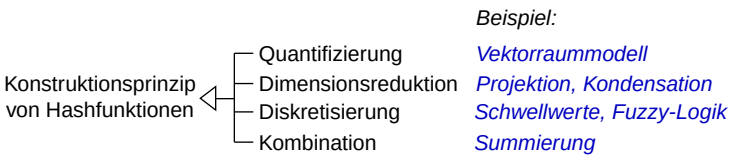

Abbildung 3.1: Die Abbildung zeit eine Taxonomie der Konstruktionsprinzipien von Hashfunktionen für das Similarity-Hashing. Exemplarisch sind die jeweils verwendeten Verfahren angegeben, die in den vorgestellten Hashfunktionen angewandt werden.

1. Wodurch können die zu hashenden Objekte quantifiziert werden?
2. Wie kann die Dimensionalität der Quantifizierung reduziert werden?
3. Wie können die reduzierten Dimensionen auf die natürlichen Zahlen abgebildet werden?
4. Wie können die Dimensionen zu einem Hashwert kombiniert werden?

Insgesamt ergibt sich aus den Antworten auf alle vier Fragen ein Prozess zum Hashen der Objekte, also eine Hashfunktion. Insbesondere die Antworten auf die ersten beiden Fragen haben Einfluss auf die Retrieval-Eigenschaften der Hashfunktion.

Eine Antwort auf die erste Frage wäre für Dokumente zum Beispiel die Aufstellung eines vektorbasierten Dokumentmodells. Dafür eignet sich zum Beispiel das Vektorraummodell, wie es in Abschnitt 1.1 definiert ist.

Würde die zweite Frage außer Acht gelassen, so wären die Hashwerte für ein Dokument sehr groß. Dies hätte zur Folge, dass ihr Platzverbrauch in der Größenordnung des Platzverbrauchs der Dokumente läge. Es ist daher notwendig, die Dimensionalität der zu hashenden Dokumente zu reduzieren. Dafür verwenden die im Folgenden vorgestellten Hashfunktionen zwei unterschiedliche Ansätze[1], nämlich die Projektion und die Kondensation der Dokumente. Unter Projektion wird allgemein die Abbildung von m_1-dimensionalen Daten in einen Raum mit geringerer Dimensionalität m_2 verstanden, wobei m_2 Dimensionen berücksichtigt werden. Die übrigen $m_1 - m_2$ Dimensionen werden vernachlässigt. Unter Kondensation verstehen wir eine Abbildung analog zur Projektion, mit dem Unterschied, dass die Dimensionen des Zielraumes jeweils eine Kombination oder Verrechnung mehrerer Dimensionen des Ursprungsraumes sind.

Die Abbildung der Vektorkomponenten reduzierter Dokumentvektoren auf natürliche Zahlen ist sinnvoll, damit der Wertebereich der Hashfunktion ein Ausschnitt der natürlichen Zahlen ist. Dazu können hier zum Beispiel Schwellwerte definiert und die dadurch auf dem Zahlenstrahl entstandenen Abschnitte durchnummeriert werden. Alternativ wurde auch die Verwen-

[1] Neben den in den Hashfunktionen verwendeten Verfahren gibt es noch zahlreiche andere. Ein vollständiger Überblick über bekannte Ansätze zur Dimensionsreduktion würde den Rahmen dieser Arbeit jedoch sprengen und nichts zum Verständnis der vorgestellten Hashfunktionen beitragen.

dung einer Zugehörigkeitsfunktion aus der Fuzzy-Logik demonstriert. Die Kombination der so gebildeten Hashwerte geschieht in den vorgestellten Hashfunktionen durch ihre Summierung.

3.2 Locality-Sensitive-Hashing

Das Rahmenwerk Locality-Sensitive-Hashing definiert eine Meta-Hashfunktion, die eine Verknüpfung zufällig gewählter Hashfunktionen aus einer Familie von Hashfunktionen ist. Sei $\mathcal{H}$ eine beliebige Hashfamilie und $h \in \mathcal{H}$ eine Hashfunktion daraus. Hashfamilien werden lokalitätssensitiv genannt, wenn die Wahrscheinlichkeit $P_{\mathcal{H}}[h(\mathbf{d}_1) = h(\mathbf{d}_2)]$, mit der die Hashwerte zweier Dokumente $\mathbf{d}_1$ und $\mathbf{d}_2$ kollidieren, beziffert werden kann und für vordefinierte Distanzen r_1 nach unten und $r_2 = r_1 \cdot (1 + \varepsilon)$ nach oben beschränkt ist.

Eine Hashfunktion h_{LSH} wird aus $\mathcal{H}$ erzeugt, indem k zufällig und unabhängig gewählte Funktionen h miteinander verknüpft werden. Der Hashwert eines Dokuments $\mathbf{d}$ entsteht durch die Kombination der einzelnen Hashwerte der gewählten Funktionen:

$$h_{LSH}(\mathbf{d}) = \psi(h_{i_1}(\mathbf{d}), \ldots, h_{i_k}(\mathbf{d})) \qquad \text{mit } h_{i_j} \in \mathcal{H}_\varphi$$

Die Kombinationsfunktion ψ bildet die Hashwerte auf $\mathbb{N}$ ab und hängt in ihrer konkreten Ausprägung von der Hashfamilie $\mathcal{H}$ ab. Die im Folgenden vorgestellte Hashfamilie verwendet hierfür beispielsweise die nachfolgende Formel, wobei z_j ein möglicher Hashwert der j-ten zufällig gewählten Hashfunktion aus dem Wertebereich $[0, \bar{z})$ ist. Die Hashwerte jeder Einzelfunktionen h_{i_j} werden komponentenweise getrennt aufsummiert und das Ergebnis als Hashwert von h_{LSH} betrachtet.

$$\psi(z_1, \ldots, z_k) = \sum_{j=1}^{k} z_j \cdot \bar{z}^{j-1} \qquad \text{mit } z_j \in [0, \bar{z})$$

Insgesamt werden l verschiedene Hashfunktionen nach diesem Schema erzeugt. Gionis u. a. (1999) haben gezeigt, dass abhängig von $P_{\mathcal{H}}[h(\mathbf{d}_1) = h(\mathbf{d}_2)]$ und dem gewählten ε Werte für k und l existieren, so dass die Wahrscheinlichkeit, mit der zwei hinreichend nahe zusammen liegende Dokumentvektoren im gleichen Bucket gespeichert werden, mindestens $P_{\mathcal{H}} \geq 1 - 1/e \approx 0.632$ beträgt. Um das zu erreichen, müssen k und l anhand der nachfolgend beschriebenen Formeln ermittelt werden.

Seien $\mathbf{d}_i$ und $\mathbf{d}_j$ zwei beliebige Dokumentvektoren. Angenommen, P_{r_1} und P_{r_2} bezeichnen die Kollisionswahrscheinlichkeiten der Hashwerte von $\mathbf{d}_i$ und $\mathbf{d}_j$ für eine zufällig aus einer lokalitätssensitiven Hashfamilie $\mathcal{H}$ gewählten Funktion h, wenn sie r_1 bzw. r_2 voneinander entfernt sind. Weiterhin sei b die maximale Anzahl von Vektoren, die in einem Bucket einer Hashtabelle gespeichert werden können. Dieser Parameter erlaubt die Anpassung von LSH an

Umgebungen mit begrenztem Speicher. n ist die Anzahl zu indizierender Dokumente.

Liegt $\mathbf{d}_j$ außerhalb der r_2-Umgebung um $\mathbf{d}_i$, so ist die Kollisionswahrscheinlichkeit für eine Meta-Hashfunktion, bestehend aus k zufälligen Funktionen aus $\mathcal{H}$, durch $P_{r_2}^k$ nach oben begrenzt. k wird so gewählt, dass die Wahrscheinlichkeit $\mathbf{d}_j$ mit $\mathbf{d}_i$ zusammen zu speichern $P_{r_2}^k = \frac{b}{n}$ beträgt. Das entspricht der Wahrscheinlichkeit, $\mathbf{d}_j$ bei einer zufälligen Auswahl dem Bucket von $\mathbf{d}_i$ zuzuordnen[2]. Es folgt:

$$k = \frac{\ln \frac{b}{n}}{\ln P_{r_2}}$$

Liegt $\mathbf{d}_j$ innerhalb der r_1-Umgebung um $\mathbf{d}_i$ ist die Kollisionswahrscheinlichkeit durch

$$P_{r_1}^k = \frac{b}{n}^{\frac{\ln P_{r_1}}{\ln P_{r_2}}}$$

nach unten begrenzt. Die Wahrscheinlichkeit, dass der Hashwert von $\mathbf{d}_j$ bei *keiner* der l Hashfunktionen h_{LSH} mit dem von $\mathbf{d}_i$ kollidiert, beträgt dann $(1 - P_{r_1}^k)^l$. Sie wird von oben genau dann durch $\frac{1}{e}$ beschränkt, wenn l wie folgt gewählt wird:

$$l = \frac{b}{n}^{-\frac{\ln P_{r_1}}{\ln P_{r_2}}}$$

Die Wahrscheinlichkeit, dass $\mathbf{d}_j$ bei wenigstens einer Hashfunktion h_{LSH} mit $\mathbf{d}_i$ kollidiert, beträgt deshalb $P_{\mathcal{H}} = 1 - \frac{1}{e}$. Das gilt für alle Hashfamilien, für die P_{r_1} und P_{r_2} beziffert werden können, falls $P_{r_1} > P_{r_2}$ gilt.

Die Parameter k und l dienen der Einstellung der Genauigkeit der Hashwerte bzw. der Anzahl der in der Meta-Hashfunktionen h_{LSH} verknüpften Funktionen aus $\mathcal{H}$. Dementsprechend dient k der Einstellung der Precision und l der Einstellung des Recalls von Locality-Sensitive-Hashing, wie in Abschnitt 2.2.3 beschrieben.

3.2.1 Hashfamilien auf Basis des Hamming-Space

Die erste Hashfamilie $\mathcal{H}_{HS}$ für Locality-Sensitive-Hashing nach Gionis u. a. (1999) basiert auf der Einbettung von Dokumentvektoren in den binären Hamming-Space. Das ist ein Vektorraum, dessen Wertebereich für Vektorkomponenten auf $\{0, 1\}$ eingeschränkt ist. Der Abstand zweier Punkte errechnet sich hier durch die Hamming-Distanz, nämlich der Anzahl unterschiedlicher Vektorkomponenten zweier Dokumentvektoren. Linial u. a. (1995) zeigen, dass zwischen eukli-

[2] Die Definition ist vergleichbar mit der von Universal-Hashing aus Abschnitt 2.2.1.

dischen Vektorräumen und dem binären Hamming-Space mit der Hamming-Distanz eine isometrische Abbildung möglich ist. Das bedeutet, dass das Verhältnis der Distanzen aller Punkte untereinander mit Hinblick auf die jeweiligen Distanzmaße bei beiden Vektorräumen gleich ist.

Sei $\mathbf{D}$ die Menge einzubettender Dokumentvektoren mit m Dimensionen und $\mathbf{d}^T = (x_1, \ldots, x_m)$ ein Vektor daraus, in dem alle $x_i \in \mathbb{N}_0$ sind[3]. Die Einbettung ermöglicht eine Familie von Hashfunktionen, für die P_{r_1} und P_{r_2} beziffert werden können:

$$\mathcal{H}_{HS} = \{h_j \mid h_j(\mathbf{d}_{HS}) = h_j((x_1, \ldots, x_m)) = x_j \quad \text{mit } j \in \{1, \ldots, m\}\}$$

Die j-te Funktion aus $\mathcal{H}_{HS}$ projiziert einen Dokumentvektor auf seine j-te Vektorkomponente. Die zufällige Auswahl einer Vektorkomponente, die in zwei Vektoren ungleich ist, hängt von ihrer Hamming-Distanz r ab und geschieht mit der Wahrscheinlichkeit $\frac{r}{m}$. Analog lassen sich für die Distanzen r_1 und r_2 dieselben Wahrscheinlichkeiten errechnen. Die Gegenwahrscheinlichkeiten sind damit die Kollisionswahrscheinlichkeiten einer zufällig aus $\mathcal{H}_{HS}$ gewählten Funktion:

$$P_{r_1} = 1 - \frac{r_1}{m}$$
$$P_{r_2} = 1 - \frac{r_2}{m}$$

Die Hashfamilie $\mathcal{H}_{HS}$ ist also lokalitätssensitiv.

3.2.2 LSH auf dünnen Daten

In den Experimenten hat sich gezeigt, dass $\mathcal{H}_{HS}$ im textbasierten Information-Retrieval nicht verwendbar ist. Werden Dokumente durch das Vektorraummodell repräsentiert, sind ihre Dokumentvektoren nur dünn besetzt. In der Praxis sind nur ca. 1% der Vektorkomponenten von 0 verschieden. Mit steigender Menge von Dokumenten sinkt der Anteil noch weiter, da in einem Dokument nur ein Bruchteil aller verfügbaren Worte verwendet werden. Diese Form von Daten heißt dünn (engl. sparse), da sie im Verhältnis zur Dimensionalität eine statistisch irrelevante Beispielmenge bildet. Die Einbettung in den Hamming-Space verschärft das Problem noch, da die Dimensionalität hier vervielfacht wird, um die reellwertigen Vektorkomponenten der Dokumentvektoren genauer abbilden zu können. Die Hamming-Distanz zwischen zwei beliebigen Dokumentvektoren wird also sehr klein, so dass alle Dokumentvektoren hier sehr nahe

[3] Wenn das nicht der Fall ist, kann der Vektorraum durch geeignete Transformationen angepasst werden, ohne dass sich die Interpunktrelationen zu stark verändern. Nach einer hinreichenden Skalierung genügt der ganzzahlige Teil reeller Vektorkomponenten für die Einbettung. Mit c wird die größte aller Vektorkomponenten aus $\mathbf{D}$ bezeichnet. Die i-te Vektorkomponente x_i wird im binären Vektor $\mathbf{d}_{HS}$ durch c Komponenten repräsentiert, x_i Einsen und $c - x_i$ Nullen. Die Dimensionalität von $\mathbf{d}_{HS}$ ist folglich $c \cdot m$.

zusammen liegen.

Für jede zufällig aus $\mathcal{H}_{HS}$ gewählte Funktion wird der Hashwert für ein beliebiges Dokument deshalb in 99% aller Fälle 0 sein. Das führt dazu, dass die Hashwerte einer Funktion h_{LSH} bei k kombinierten Hashfunktionen mit 0.99^k gleich 0 ist. Es müssten daher sehr große Werte für k und l gewählt werden, um eine sinnvolle Indizierung aller Dokumentvektoren zu ermöglichen. Angenommen, die durchschnittliche Distanz eines Dokumentvektors im Hamming-Space zu seinem nächsten Nachbarn betrüge $r_1 = 10$ bei einer Dimensionalität $m = 10000$ und einer Menge von $n = 1000$ Dokumenten. Sei außerdem die Größe eines Buckets $b = 10$ und $\varepsilon = 0.1$. Es folgt $r_2 = 11$ als akzeptable ε-Umgebung. Daraus ergeben sich die Werte $k = 4184$ und $l = 65$. Die Berechnung der Hashwerte für alle Dokumente mithilfe von $\mathcal{H}_{HS}$ würde einen unverhältnismäßig hohen Aufwand bedeuten. Es ist zu beachten, dass in der Praxis die Werte n und m noch deutlich größer sind.

Dieses Verhalten ist anhand der eingesetzten Distanzmaße für Hamming- bzw. Dokumentvektoren zu erklären. Die Hamming-Distanz basiert auf der Berechnung der *Unterschiede* zwischen zwei Vektoren. Da sich Dokumentvektoren im Allgemeinen aber nur in wenigen Vektorkomponenten tatsächlich voneinander unterscheiden, und die übrigen größtenteils mit 0 beziffert werden, muss Locality-Sensitive-Hashing hier besonders sensibel eingestellt werden, damit auch verhältnismäßig kleine Unterschiede eine Auswirkung auf die Wahrscheinlichkeit für eine Hashkollision zweier Dokumente hat. In dieser Domäne wäre daher eine Hashfunktion interessant, die auf Ähnlichkeits- oder Distanzmaßen basiert, in denen ausschließlich überlappende Gemeinsamkeiten berücksichtigt werden, wie zum Beispiel dem Jaccard-Koeffizienten, der Kosinusähnlichkeit oder der Minkowski-Distanz.

3.2.3 Hashfamilien auf Basis α-stabiler Wahrscheinlichkeitsverteilungen

Eine Hashfamilie $\mathcal{H}_\alpha$, die auf dünne Daten anwendbar ist, wurde von Datar u. a. (2004) vorgeschlagen. Das Konzept der Hashfamilie ist einfach; das Skalarprodukt $\mathbf{a}^T \cdot \mathbf{d}$ zwischen einem Vektor $\mathbf{a}$, dessen Vektorkomponenten reelle Zufallszahlen sind, und einem Dokumentvektor $\mathbf{d}$ dient als Reduktionsschritt in Form einer Kondensation. Je kleiner die Differenz der Skalarprodukte von $\mathbf{a}$ mit zwei Dokumentvektoren ist, desto ähnlicher sind sich diese. Die Abbildung eines Skalarprodukts auf eine natürliche Zahl gelingt durch die Aufteilung seines Wertebereichs $\mathbb{R}$ in gleich große Abschnitte. Die Abschnitte werden durchnummeriert und jede Abschnittsnummer dient als Hashwert. Auf diese Weise werden hinreichend ähnliche Dokumentvektoren derselben Abschnittsnummer zugeordnet.

Das Ganze basiert auf so genannten α-stabilen Wahrscheinlichkeitsverteilungen. Diese Form von Verteilungen hat die besondere Eigenschaft, summenstabil zu sein. Die bekannteste Wahrscheinlichkeitsverteilung mit dieser Eigenschaft ist die Gauss-Verteilung. Für zwei Gauss-ver-

teilte Zufallsvariablen gilt, dass ihre Summe ebenfalls Gauss-verteilt ist. Eine ausführliche Einführung in die Theorie um α-stabile Verteilungen gibt Nolan (2005).

Eine Wahrscheinlichkeitsverteilung heißt α-stabil für $\alpha \geq 0$, wenn für die unabhängig, identisch verteilten Zufallsvariablen $X_0, X_1, \ldots, X_m$ und die reellen Zahlen $x_1, \ldots, x_m$ gilt ($\sim$ bedeutet „ist genauso verteilt wie“):

$$\sum_{i=1}^{m} x_i X_i \sim \left(\sum_{i=1}^{m} {x_i}^{\alpha} \right)^{\frac{1}{\alpha}} X_0$$

Für alle $\alpha \in (0, 2]$ existieren Verteilungen, die diese Bedingung erfüllen. Es können jedoch für fast alle möglichen Werte keine Dichtefunktionen angegeben werden. Die einzigen Werte für α, für die das möglich ist, sind $\frac{1}{2}, 1$ und 2, nämlich die Standard-Lévy-Verteilung, die Standard-Cauchy-Verteilung bzw. die Standard-Gauss-Verteilung:

$$f_{\frac{1}{2}}(x) = \sqrt{\frac{1}{2\pi}} \cdot \frac{1}{x^{3/2}} \cdot \exp\left(-\frac{1}{2x}\right)$$
$$f_1(x) = \frac{1}{\pi(1+x^2)}$$
$$f_2(x) = \frac{1}{\sqrt{2\pi}} \cdot \exp\left(-\frac{x^2}{2}\right)$$

Eine der Anwendungen α-stabiler Verteilungen ist die Abschätzung des Abstands zwischen zwei hochdimensionalen (Dokument-)Vektoren $\mathbf{d}_1$ und $\mathbf{d}_2$ nach Indyk (2000): Sei $\mathbf{a}$ ein m-dimensionaler Vektor, bei dem jede Komponente unabhängig, identisch verteilt aus einer α-stabilen Verteilung gewählt wurde. Sei ferner $\mathbf{d}_i^T = (x_1, \ldots, x_m)$ ein Vektor, dann gilt für das Skalarprodukt beider Vektoren:

$$\mathbf{a}^T \cdot \mathbf{d}_i \sim \left(\sum_j x_j^{\alpha} \right)^{\frac{1}{\alpha}} X = \|\mathbf{d}_i\|_{\alpha} X$$

X ist α-stabil verteilt und $\|\mathbf{d}_i\|_{\alpha}$ ist die L_α-Norm von $\mathbf{d}_i$. Mithilfe einer kleinen Anzahl verschiedener Vektoren $\mathbf{a}$ können die Mengen von Skalarprodukten $\mathbf{a}^T \cdot \mathbf{d}_i$ für $i \in \{1, 2\}$ dazu verwendet werden, den Abstand $\|\mathbf{d}_1 - \mathbf{d}_2\|_{\alpha}$ mit hoher Wahrscheinlichkeit annähernd korrekt abzuschätzen.

Das Skalarprodukt ist distributiv, es gilt also $\mathbf{a}^T \cdot \mathbf{d}_1 - \mathbf{a}^T \cdot \mathbf{d}_2 = \mathbf{a}^T(\mathbf{d}_1 - \mathbf{d}_2)$. Daraus folgt:

$$\mathbf{a}^T \cdot \mathbf{d}_1 - \mathbf{a}^T \cdot \mathbf{d}_2 \sim \|\mathbf{d}_1 - \mathbf{d}_2\|_{\alpha} X$$

Der Abstand $\|\mathbf{d}_1 - \mathbf{d}_2\|_{\alpha}$ bedingt also die Wahrscheinlichkeit, dass die Skalarprodukte $\mathbf{a}^T \cdot \mathbf{d}_1$

und $\mathbf{a}^T \cdot \mathbf{d}_2$ annähernd gleich groß sind. Je geringer der Abstand, desto größer ist die Wahrscheinlichkeit.

Wird der Zahlenstrahl reeller Zahlen in gleichmäßige Segmente der Breite $u > 0$ unterteilt und jedem Segment ein Hashwert aus $\mathbb{N}$ zugewiesen, lässt sich eine lokalitätssensitive Hashfamilie $\mathcal{H}_\alpha$ wie folgt definieren:

$$\mathcal{H}_\alpha = \{h_{\mathbf{a},c} \mid \forall \mathbf{a} : h_{\mathbf{a},c}(\mathbf{d}) = \left\lfloor \frac{\mathbf{a}^T \cdot \mathbf{d} + c}{u} \right\rfloor \text{ mit } c \in [0, u]\}$$

Die Wahrscheinlichkeit P_{r_1} der Kollision der Hashwerte zweier Vektoren $\mathbf{d}_1$ und $\mathbf{d}_2$ abhängig von ihrer Distanz $r = ||\mathbf{d}_1 - \mathbf{d}_2||_\alpha$ ist mithilfe der Dichtefunktion der α-stabilen Verteilung $f_\alpha(x)$ berechenbar:

$$P_{r_1} = \int_0^u \frac{1}{r_1} \cdot f_\alpha\left(\frac{x}{r_1}\right) \cdot \left(1 - \frac{x}{u}\right) dx$$

Analog errechnet sich P_{r_2}; $\mathcal{H}_\alpha$ ist also lokalitätssensitiv.

3.2.4 Aktuelle Entwicklungen

Für die Idee des Similarity-Hashing existieren zahlreiche Anwendungsfälle, insbesondere für Hashfamilien des LSH-Rahmenwerks, die seine Anwendbarkeit in den unterschiedlichsten Fachgebieten demonstrieren. Darunter ist ein Ansatz zur Clusteranalyse einer Menge von Webseiten (Haveliwala u. a. 2000), das Data-Mining nach DNA-Sequenzen oder molekularen Strukturen in großen Korpora (Buhler 2001; Dutta u. a. 2006) sowie das Erkennen von Posen und Handzeichen in Videos (Shakhnarovich u. a. 2003).

Darüber hinaus haben Andoni und Indyk (2006) eine neue Familie von Hashfunktionen vorgestellt, die auf das Pattern-Matching-Problem anwendbar ist. Hier geht es darum, in einem Text nach verschiedenen Pattern, also zum Beispiel kurzen Zeichenketten, zu suchen. Eine weitere Arbeit von Bawa u. a. (2005) entwickelt das Rahmenwerk weiter, so dass der Parameter k, also die Anzahl zufällig gewählter Hashfunktionen, nicht mehr benötigt wird. Stattdessen wird für jedes Dokument einzeln entschieden, wie viele Hashfunktionen miteinander verknüpft werden. Die dafür vorgeschlagene Datenstruktur LSH-Forest ist ein binärer Baum. Es wird vorausgesetzt, dass jede Funktion der eingesetzten Hashfamilie den Wertebereich $\{0, 1\}$ hat. Der Hashwert eines Dokuments ergibt sich dann aus der Verknüpfung der Kantenbeschriftungen auf einem Pfad von der Wurzel zu einem Blatt, abhängig von den Hashwerten der verwendeten Hashfunktionen.

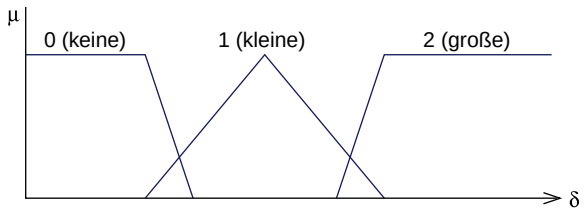

Abbildung 3.2: Ein Fuzzyfizierungsschema mit 3 Intervallen, wovon jedes für eine der linguistischen Variablen „keine-“, „kleine-“ oder „große Abweichung“ steht. Jede der Variablen wird durch eine der Zahlen $0, 1, 2$ repräsentiert. Ein konkreter Abweichungswert δ wird mit dem Schema auf eine oder mehrere der linguistischen Variablen abgebildet. Bei Überlappung wird immer genau eine der Möglichkeiten gewählt.

3.3 Fuzzy-Fingerprinting

Die Einbeziehung von Domänenwissen des textbasierten Information-Retrieval ist die Grundlage der Familie von Hashfunktionen im Fuzzy-Fingerprinting (Stein 2005a). Die Idee dabei ist (1.), eine sehr kleine Menge von diskriminativen Charakteristika zur Quantisierung von Dokumenten zu verwenden und sie (2.) in Beziehung zu einem erwarteten Wert dafür zu setzen, bzw. sie damit zu normalisieren. Die Charakteristika sind das Ergebnis einer Kondensation der Dimensionen eines Dokumentvektors. Mithilfe einer Zugehörigkeitsfunktion, wie sie in der Fuzzy-Logik verwendet wird, lassen sich die Abweichungen der Charakteristika eines Dokuments von ihren Erwartungswerten auf so genannte linguistische Variablen (z.B. „keine“, „kleine“ oder „große“ Abweichung) abbilden. Jede linguistische Variable wird durch eine natürliche Zahl repräsentiert, so dass aus ihnen ein Hashwert kombiniert werden kann. Dieses Prinzip spannt eine neue Dimension von Hashfunktionen auf, die *kontextsensitiv* sind. Das heißt, es lässt sich abseits der betrachteten Domäne auch auf andere übertragen; der kritische Aspekt liegt hier in der Wahl oder Konstruktion sinnvoller Charakteristika.

Angenommen, ein Dokument wird durch m Charakteristika $C_1, \ldots, C_m$ repräsentiert, so dass $\mathbf{d}^T = (x_1, \ldots, x_m)$ als Dokumentmodell[4] dient, bei dem C_i durch den i-ten Wert x_i beziffert wird. Mit $E[C_i]$ wird der Erwartungswert von C_i und mit $\delta_i = |1 - \frac{x_i}{E[C_i]}|$ die normierte Abweichung davon bezeichnet. Es ergibt sich ein Vektor $\Delta^T = (\delta_1, \ldots, \delta_m)$. Sei μ die Zugehörigkeitsfunktion eines Fuzzyfizierungsschemas mit $|\mu|$ Intervallen, wie es in Abbildung 3.2 zu sehen ist. $\mu(\delta_i) \in \{0, \ldots, |\mu| - 1\}$ ist der fuzzyfizierte Betrag von δ_i, so dass ein Vektor $\Delta_\mu^T = (\mu(\delta_1), \ldots, \mu(\delta_m))$ entsteht. Die hierfür vorgeschlagene Hashfunktion lautet:

$$h_\mu(\mathbf{d}) = \sum_{i=1}^{m} \mu(\delta_i) \cdot |\mu|^{i-1}$$

[4] Diese Definition ist sehr ähnlich zu der des Vektorraummodells, in dem die Wichtigkeit jedes Wortes in Bezug auf ein Dokument als Charakteristikum dient. Dennoch ist dieses Modell für Fuzzy-Fingerprinting ungeeignet, da das erwünschte Ziel einer kleinen Anzahl von Dimensionen nicht gegeben ist.

3.3.1 Hashing von Dokumenten mit Fuzzy-Fingerprinting

Um Dokumente mithilfe von Fuzzy-Fingerprinting zu hashen, ist die Definition einer kleinen Anzahl von Charakteristika notwendig, die geeignet sind, ein Dokument von anderen abzugrenzen. Voraussetzung dabei ist, dass die erwarteten Werte jedes Charakteristikums bekannt sind oder experimentell ermittelt werden können.

Die von Stein (2005a) vorgeschlagenen Charakteristika sind so genannte Präfixäquivalenzklassen (kurz: Präfixklassen). Eine Präfixklasse $\mathcal{P}_\omega$ für eine Buchstabenfolge ω ist die Menge aller Worte $W_\omega \subseteq W$, die mit ω beginnen. ω ist Präfix aller Worte aus W_ω. Darauf aufbauend ist auch die Kombination verschiedener Präfixklassen denkbar, so dass $\mathcal{P}_{\omega_1,\omega_2} = \mathcal{P}_{\omega_1} \cup \mathcal{P}_{\omega_2} = W_{\omega_1} \cup W_{\omega_2}$ gilt.

Um Dokumente durch verschiedene Präfixklassen zu repräsentieren, ist eine Gewichtung jeder Klasse notwendig. Im einfachsten Fall ist das die Anzahl der Worte eines Dokuments, die einer Präfixklasse angehören. Analog zum Vektorraummodell sollte der Vektor $\mathbf{d}$ aber normiert werden, so dass lange Dokumente gegenüber kurzen keine zu starke Gewichtung je Präfixklasse erhalten. Des Weiteren sollte insbesondere bei Verwendung kombinierter Präfixklassen darauf geachtet werden, dass bestimmte Präfixe nicht häufiger vertreten sind als andere. Das entspräche gegenüber den übrigen Präfixen einer Vervielfachung ihres Gewichts.

Eine Menge verwendbarer Präfixe ist zum Beispiel das lateinische Alphabet. Jeder Buchstabe dient dabei als Präfix einer Präfixklasse, so dass Dokumente durch einen 26-dimensionalen Vektor repräsentiert werden. Die Erwartungswerte $E[\mathcal{P}_\omega]$ mit $\omega \in \{a, \ldots, z\}$ lassen sich anhand eines hinreichend großen Korpus von Dokumenten ermitteln. Der Korpus sollte so gewählt werden, dass eine möglichst repräsentative Auswahl an Texten einer Sprache oder eines speziellen Themengebiets analysiert wird. Abbildung 3.3 zeigt exemplarisch die Erwartungswerte der oben genannten Präfixklassen.

Das Präfixklassenmodell lässt sich auf zwei Arten variieren; (1.) über die Präfixlänge $|\omega|$ der Präfixklassen $\mathcal{P}_\omega$ und (2.) durch die Vereinigung verschiedener Präfixklassen zu kombinierten Präfixklassen. Offensichtlich erlauben beide Varianten die Einstellung der Genauigkeit der Hashwerte, die mit der Hashfunktion berechnet werden. Die Wahrscheinlichkeit, mit der die Hashwerte zweier (beliebiger) Dokumente kollidieren, ist davon abhängig. Je kleiner der Wertebereich ist, desto größer wird sie. Indirekt wird dadurch die Konfiguration der Hashfunktion zur Beeinflussung von Precision und Recall ermöglicht.

Die erste Variante entspricht der Annäherung des Präfixklassenmodells an das Vektorraummodell. Mit steigender Präfixlänge $|\omega|$ nähern sich die Präfixe Worten an, so dass ab einer bestimmten Länge eine Präfixklasse höchstens ein Wort enthält. Die Anzahl der Präfixklassen und damit auch die Genauigkeit der Hashwerte steigt hier exponentiell. Für die Präfixlänge $|\omega|$ ergeben sich $26^{|\omega|}$ Präfixklassen. Da nicht jeder Präfix der Länge $|\omega| > 1$ tatsächlich Präfix eines oder

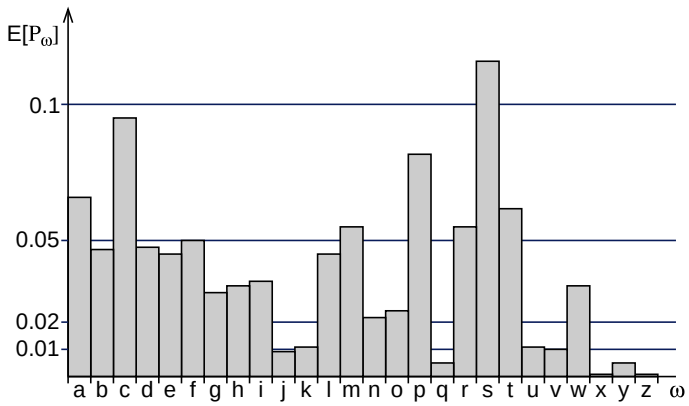

Abbildung 3.3: Die Grafik zeigt die erwartete Häufigkeit $E[\mathcal{P}_\omega]$ der Präfixklasse des Präfixes ω in einem englischsprachigen Dokument. Die Beträge wurden durch Auswertung des „British National Corpus (BNC)“ ermittelt (Burnard 2000). Der Korpus setzt sich aus geschriebenen und gesprochenen Beispielen des Englischen zusammen. Mit einer Größe von insgesamt etwa 100 Millionen Worten bietet er einen repräsentativen Querschnitt.

mehrerer Worte ist[5], und infolgedessen die erwartete Häufigkeit ihres Auftretens 0 ist, können sie bei der Aufstellung eines Präfixklassenmodells ignoriert werden.

Die zweite Variante ermöglicht die Einstellung einer festen Anzahl k von Präfixklassenkombinationen und damit eine genaue Abstufung zwischen den beiden Präfixklassenmodellen der Präfixlängen $|\omega| = i$ bzw. $|\omega| = i + j$ mit $26^i \leq k < 26^{i+j}$ und $i, j > 0$. Die Aufstellung von k kombinierten Präfixklassen gelingt durch Unterteilung aller 26^{i+j} Präfixklassen in k disjunkte Teilmengen. Jede Teilmenge wird als kombinierte Präfixklasse $\mathcal{P}_{\cup_i \omega_i}$ aufgefasst, deren erwartete Häufigkeit sich aus der Summe erwarteter Häufigkeiten der enthaltenen Präfixklassen ergibt:

$$E[\mathcal{P}_{\cup_i \omega_i}] = \sum_i E[\mathcal{P}_{\omega_i}]$$

Es stellt sich die Frage, wie für ein gegebenes k eine geeignete Unterteilung der übergeordneten Menge von Präfixklassen gefunden werden kann. Eine zufällige Aufteilung ist nicht sinnvoll, da hier auch Fälle möglich sind, in denen eine kombinierte Präfixklasse die Mehrzahl aller Präfixklassen vereint und die $k - 1$ übrigen jeweils nur wenige enthalten. Wir schlagen deshalb eine Aufteilung der Präfixklassen abhängig von ihren erwarteten Häufigkeiten vor, so dass die kumulierten erwarteten Häufigkeiten der k kombinierten Präfixklassen in etwa gleich hoch sind. Wenn jede kombinierte Präfixklasse als Behälter der Größe $1/k$ aufgefasst wird und jede Präfixklasse eine „Masse“ entsprechend ihrer erwarteten Häufigkeit hat, wird deutlich, dass diese Problemstellung eine Instanz des Behälterproblems ist, wobei k die optimale Anzahl von Behältern ist. Das Behälterproblem ist aus der theoretischen Informatik bekannt und zählt zu den NP-vollständigen Problemen. Es ist also bis dato kein Algorithmus bekannt, der eine optimale Lösung des Problems in polynomieller Zeit errechnet.

[5] Im Englischen betrifft das für $|\omega| = 2$ unter anderen die Präfixe „vb“, „xb“, „xc“ und „yc“.

Stattdessen wurden verschiedene Auswahlheuristiken vorgeschlagen, mit denen eine annähernd optimale Lösung errechnet werden kann. Der Algorithmus arbeitet dazu die Präfixklassen sequentiell ab. Für jede Präfixklasse wird mit der Heuristik eine kombinierte Präfixklasse gewählt, der sie zugeordnet wird. Falls keine der bisher vorhandenen kombinierten Präfixklassen genug Platz bietet, wird eine neue erzeugt. Die bekanntesten Auswahlheuristiken heißen Next-Fit, First-Fit und First-Fit-Decreasing. Die ersten beiden arbeiten wie folgt: Eine Präfixklasse wird entweder der nächstmöglichen kombinierten Präfixklasse, ausgehend von der zuletzt betrachteten, oder aber der erstmöglichen kombinierten Präfixklasse, ausgehend von der zuerst erzeugten, zugeordnet. First-Fit-Decreasing verfolgt dieselbe Strategie wie First-Fit, die Präfixklassen werden zuvor jedoch absteigend nach ihren erwarteten Häufigkeiten sortiert. Damit wird eine Aufteilung möglich, die höchstens $\frac{11}{9} \cdot k + 4$ kombinierte Präfixklassen erzeugt, wobei k die optimale Anzahl ist.

Um nicht mehr als k kombinierte Präfixklassen zu erzeugen, werden die über k hinausgehenden Präfixklassen in einem Nachverarbeitungsschritt auf die vorigen kombinierten Präfixklassen verteilt. Eine einfache Verteilungsstrategie könnte zum Beispiel auf einer zufälligen Auswahl basieren.

Kapitel 4

Evaluierung der Hashfunktionen

Dieses Kapitel beinhaltet die Evaluierung der in Kapitel 3 vorgestellten Hashfunktionen. Die durchgeführten Experimente dienen der Beantwortung folgender Fragesten:

- Wie verhalten sich die Retrieval-Eigenschaften von Locality-Sensitive-Hashing zu denen von Fuzzy-Fingerprinting?
- Wie wirkt sich die Einbeziehung von Domänenwissen auf die Retrieval-Eigenschaften der Indizierung im Similarity-Hashing aus?
- Wie stark vergrößert sich die durchschnittliche Anzahl gefundener Dokumente mit der Anzahl indizierter Dokumente?
- Wie gut eignen sich Präfixklassen als Grundlage für ein Dokumentmodell im Vergleich zum Vektorraummodell?

Das Kapitel unterteilt sich in drei Abschnitte. Der erste Abschnitt beschäftigt sich mit der Beschaffenheit von Korpora, auf deren Grundlage die Experimente durchgeführt worden sind. Im zweiten und dritten Abschnitt werden die Ergebnisse der Experimente diskutiert.

4.1 Korpora und Testkollektionen für die Experimente

Die Experimente basieren auf drei verschiedenen Dokumentkorpora. Dabei handelt es sich um veröffentlichte Sammlungen von Dokumenten, die unter anderem dem Zweck dienen, für dritte nachvollziehbare Experimente durchzuführen. Jede der drei enthält mehrere tausend englischsprachige Dokumente aus verschiedenen realitätsnahen Quellen. Die Dokumente aus einem

Korpus sind zusätzlich in thematische Kategorien einsortiert, was insbesondere für die Beantwortung der drittgenannten Fragestellung notwendig ist.

Der „Reuters Corpus Volume 1“ (RCV1) nach Lewis u. a. (2004) ist ein kategorisierter Korpus und besteht aus über 800.000 Meldungen des Reuters Nachrichtendienstes, die manuell um Metainformationen, wie Kategorie, geographische Region oder Industriesektor angereichert wurden. RCV1 unterteilt sich in insgesamt 103 verschiedene Kategorien, die hierarchisch auf vier Hauptkategorien verteilt sind. Jede Unterkategorie verfeinert die von ihrer Oberkategorie bezeichneten Themengebiete. Ein Dokument kann mehreren Kategorien verschiedener Ebenen zugeordnet sein, für die Experimente ist jedoch eine eindeutige Zuordnung notwendig. Bei der Zusammenstellung von Testkollektionen wurden deshalb nur Dokumente berücksichtigt, deren spezifischste Kategorie sich nicht von der spezifischsten Kategorie eines anderen Dokumentes ableitet. Zwei Dokumente gehören zu einer gemeinsamen Kategorie, wenn sie sowohl die Hauptkategorie als auch die spezifischste Kategorie gemein haben.

Der Plagiatkorpus nach Kulig (2006) ist eine Sammlung künstlich generierter Dokumente. Zweck des Korpus’ ist, verschiedene Formen des Plagiarismus zu simulieren. Als Plagiat wird im Allgemeinen das Zitat der Ideen, Texte oder Erfindungen anderer verstanden, ohne eine entsprechende Kennzeichnung vorzunehmen. Der Korpus basiert auf wissenschaftlichen Papieren, die der „ACM Digital Library“ entnommen wurden. Der Text der Papiere wurde extrahiert und abschnittsweise indiziert. Aus den so präparierten Dokumenten konnten dann, anhand eines Konstruktionsalgorithmus, Instanzen verschiedener Formen des Plagiarismus generiert und zu Dokumenten zusammengefasst werden. Der für diese Ausarbeitung generierte Korpus enthält insgesamt 3.000 Dokumente.

Die „Request For Comments Collection“ (Postel 2006) ist eine Sammlung aller ca. 4.500 Dokumente über standardisierte Internettechnologien. Dieser Korpus hat die besondere Eigenschaft, dass viele Dokumente sich überschneiden. Das heißt, dass mitunter viele Passagen denselben oder einen ähnlichen Wortlaut haben, da die Dokumente häufigen Revisionen unterliegen. Die beiden letztgenannten Korpora erlauben daher die Evaluierung der Hashfunktionen unter der Prämisse, dass viele Dokumente große Ähnlichkeit aufweisen.

Für die Experimente wurden auf Grundlage der Korpora zahlreiche Testkollektionen verschiedener Größen mit zufällig gewählten Dokumenten zusammengestellt. Die im Einzelnen verwendeten Zusammenstellungen werden zusammen mit den jeweiligen Experimenten beschrieben.

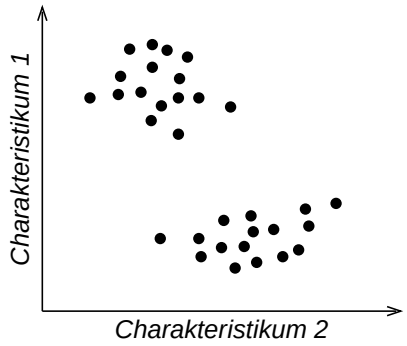

Abbildung 4.1: Punkte in einem zweidimensionalen Raum. Eine Clusteranalyse auf der Menge aller dargestellten Punkte identifiziert im Idealfall zwei Cluster von Punkten. Das menschliche Auge identifiziert hier direkt die einzig sinnvolle Unterteilung.

4.2 Dokumentmodelle für Fuzzy-Fingerprinting versus Vektorraummodell

Die Bewertung von Dokumentmodellen geschieht im Information-Retrieval häufig im Umfeld des konkreten Anwendungsfalls der Clusteranalyse und ihrem Abschneiden darin. Die Clusteranalyse ist ein Verfahren des Data-Mining und beschäftigt sich mit der Erkennung von Strukturen, so genannte Cluster, in mehrdimensionalen Daten. Werden die Dokumente einer Kollektion $\mathbf{D}$ durch mehrdimensionale Vektoren repräsentiert, entsteht durch die Definition einer Ähnlichkeits- oder Distanzfunktion darauf ein Vektorraum. Die Dokumentvektoren verteilen sich in der Praxis nicht gleichmäßig über diesen Raum, so dass ein Vektor unter Umständen zusammen mit anderen eine „Wolke“ bildet, die von anderen Wolken durch einen leeren Raumabschnitt getrennt ist. Abbildung 4.1 illustriert diesen Fall. Ziel der Clusteranalyse ist es, solche Wolken zu erkennen und damit ähnliche Dokumente einander zuzuordnen.

Eine andere Anwendung der Clusteranalyse im Information-Retrieval ist die Nachbearbeitung der Treffer einer Suchmaschine für eine Anfrage. Alle großen Suchmaschinen liefern heutzutage eine Liste der Treffer, die nach Relevanz sortiert ist. Der Nutzer der Suchmaschine hat dann die Aufgabe, die für ihn tatsächlich relevanten Dokumente aus der Liste zu extrahieren. Die Clusteranalyse der Suchergebnisse ermöglicht dagegen ihre inhaltliche und gleichsam thematische Aufteilung in Untergruppen. Zusammen mit Verfahren zur Generierung einer Beschreibung für jeden Cluster bietet sich dem Nutzer die Möglichkeit, die Suchergebnisse zuerst nach für ihn relevanten und irrelevanten Themen zu trennen[1], bevor er einzelne Treffer betrachtet.

In Appendix A werden sowohl Algorithmen zur Clusteranalyse, als auch Methoden zur Bewertung ihrer Ergebnisse vorgestellt. Alle dort diskutierten Verfahren sind in den Experimenten bzw. zu ihrer Auswertung verwendet worden.

[1] Suchmaschinen dieser Machart sind zum Beispiel AISearch, Vivísimo und Clusty.

Präfixmodell mit variablem $\|\omega\|$	F-Measure-Abweichungen min.	max.	∅
VSM	— Basislinie —		
$\|\omega\| = 1$	-66%	-52%	-59%
$\|\omega\| = 2$	-62%	-46%	-56%
$\|\omega\| = 3$	-60%	-51%	-55%

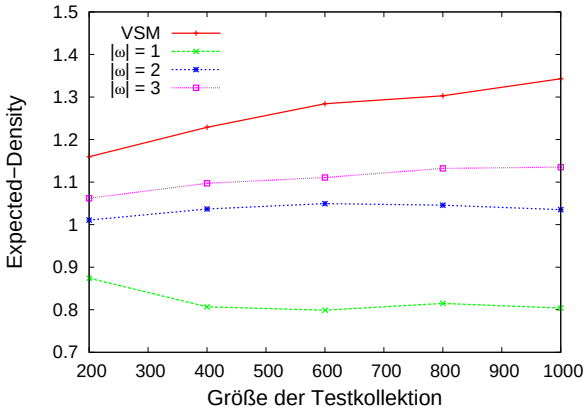

Abbildung 4.2: Ergebnisse der Experimente zum Vergleich des Vektorraummodells mit verschiedenen einfachen Präfixklassenmodellen für verschiedene Präfixlängen $|\omega|$. Die Tabelle zeigt jeweils die minimalen, maximalen und durchschnittlichen Abweichungen der F-Measure-Werte von der Basislinie, die aus den Einzelergebnissen der Clusteralgorithmen MajorClust, K-Means und Group-Average-Link gemittelt wurden. In der Grafik sind die korrespondierenden $\bar{\rho}$-Werte zu sehen. Die Experimente basieren auf Testkollektionen aus dem RCV1 der Größen 200 bis 1000 Dokumente (in 200er-Schritten) aus 5 verschiedenen Kategorien. Für jede Kollektionsgröße wurden insgesamt 10 zufällige Zusammenstellungen generiert, so dass insgesamt 50 Kollektionen verwendet wurden.

4.2.1 Analyse des Präfixklassenmodells

Zur Analyse der verschiedenen Varianten des Präfixklassenmodells wurden die Expected-Density-Werte der mit ihnen erstellten Ähnlichkeitsgraphen ermittelt und eine Reihe von Clusteranalysen auf den Testkollektionen durchgeführt. Als Basislinie der Experimente dient das klassische Vektorraummodell. Für die erste Variante, der Veränderung der Präfixlängen $|\omega|$, wurden die drei Präfixklassenmodelle mit jeweils allen Präfixklassen für $|\omega| \in \{1, 2, 3\}$ verwendet. Abbildung 4.2 zeigt die Ergebnisse.

Das Präfixklassenmodell ist generell schlechter geeignet, Dokumente zu repräsentieren, als das Vektorraummodell, da zu keiner Zeit positive F-Abweichungen bzw. bessere $\bar{\rho}$-Werte ermittelt wurden. Das entspricht den Erwartungen, da das Präfixklassenmodell gegenüber dem Vektorraummodell einen deutlich höheren Abstraktionsgrad hat. Es ist zu erkennen, dass sich die F-Abweichungen mit steigenden Präfixlängen mehr und mehr der Basislinie annähern. Auch die $\bar{\rho}$-Werte nähern sich dem des Vektorraummodells mit steigender Präfixlänge von unten an. Je länger die Präfixe der eingesetzten Präfixklassen sind, desto genauer bildet das entsprechende Präfixklassenmodell Dokumente ab.

Die zweite Variationsmöglichkeit von Präfixklassenmodellen, die Kombination verschiedener Präfixklassen mit dem Ziel, eine feste Anzahl k kombinierter Präfixklassen aufzustellen, wurde anhand des in Abschnitt 3.3.1 vorgeschlagenen Verfahrens analysiert. Für k wurden Werte aus $\{13, 26, 100, 676\}$ verwendet. Insbesondere der direkte Vergleich der Modelle aus 26 und 676 kombinierten Präfixklassen mit den eingangs getesteten Modellen aus einfachen Präfixklassen ist von Interesse. Zusätzlich erlauben die Experimente mit Modellen aus 13 und 100 kombinier-

Präfixmodell mit variablem k	F-Measure-Abweichungen min.	max.	∅
VSM		— Basislinie —	
$k = 13$	-65%	-56%	-62%
$k = 26$	-62%	-45%	-54%
$k = 100$	-50%	-26%	-40%
$k = 676$	-41%	-10%	-25%

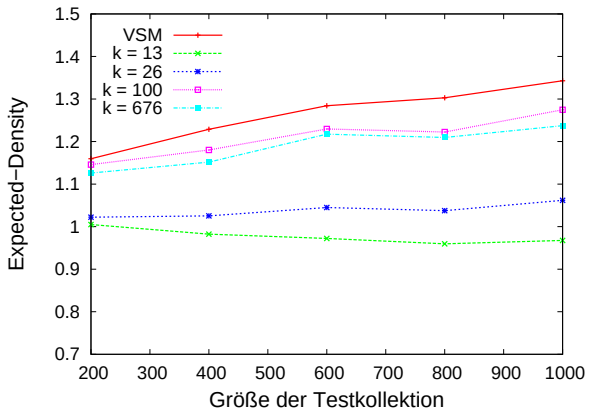

Abbildung 4.3: Ergebnisse der Experimente zum Vergleich des Vektorraummodells mit Präfixklassenmodellen aus k kombinierten Präfixklassen mit gleichmäßig verteilten erwarteten Häufigkeiten. Die Tabelle zeigt jeweils die minimalen, maximalen und durchschnittlichen Abweichungen der F-Measure-Werte von der Basislinie, die aus den Einzelergebnissen der Clusteralgorithmen MajorClust, K-Means und Group-Average-Link gemittelt wurden. In der Grafik sind die korrespondierenden $\bar{\rho}$-Werte zu sehen. Die Experimente basieren auf Testkollektionen aus dem RCV1 der Größen 200 bis 1000 Dokumente (in 200er-Schritten) aus 5 verschiedenen Kategorien. Für jede Kollektionsgröße wurden insgesamt 10 zufällige Zusammenstellungen generiert, so dass insgesamt 50 Kollektionen verwendet wurden.

ten Präfixklassen eine Aussage darüber, wie sie sich mit fallendem bzw. steigendem k verhalten. Die Ergebnisse der Experimente sind in Abbildung 4.3 zu sehen.

Bemerkenswert ist, dass Präfixklassenmodelle mit kombinierten Präfixklassen bessere Retrieval-Eigenschaften aufweisen, als Präfixklassenmodelle für verschiedene Präfixlängen. Die F-Abweichungen des Modells aus $k = 26$ kombinierten Präfixklassen sind besser als die korrespondierenden Abweichungen für $|\omega| = 1$. Bei den $\bar{\rho}$-Werten ist der Unterschied noch auffälliger, denn hier schneidet auch das Modell aus 13 kombinierten Präfixklassen besser ab. Folglich hat die Kombination von Präfixklassen und die damit verbundene Vereinheitlichung ihrer erwarteten Häufigkeiten einen positiven Effekt auf das Präfixklassenmodell.

In der Summe sind folgende Schlussfolgerungen zu ziehen:

- Präfixklassenmodelle aus kombinierten Präfixklassen weisen bessere Retrieval-Eigenschaften auf, als Präfixklassenmodelle aus einfachen Präfixklassen.
- Algorithmisch sind beide Varianten nahezu gleichwertig. Die kombinierten Präfixklassen müssen für jedes k nur einmal errechnet werden.
- Der Einsatz des Modells aus kombinierten Präfixklassen erlaubt bei Fuzzy-Fingerprinting — analog zu Locality-Sensitive-Hashing sowie abhängig von k — die exakte Einstellung der Genauigkeit der Hashwerte. Dies ist, neben den Fuzzifizierungsschemata, ein weiteres Mittel zur Beeinflussung von Precision und Recall der Hashfunktionen.

4.3 Locality-Sensitive-Hashing versus Fuzzy-Fingerprinting

Die Gegenüberstellung der Rahmenwerke Locality-Sensitive-Hashing und Fuzzy-Fingerprinting erfolgt anhand des Anwendungsfalls der Indizierung einer Menge von Dokumenten. Konkret wurden Hashfunktionen basierend auf α-stabilen Wahrscheinlichkeitsverteilungen als Vertreter für LSH und Hashfunktionen basierend auf einfachen und kombinierten Präfixklassenmodellen als Vertreter für FF eingesetzt.

Als Vergleichsmaßstab für die Hashfunktionen dient die Analyse der Retrieval-Eigenschaften der Hashfunktionen, also die durchschnittliche Precision und der durchschnittliche Recall pro Suchanfrage. Für die Ermittlung beider Werte ist die Einteilung gefundener Dokumente in relevante und irrelevante notwendig. Das geschieht anhand einer Reihe von Ähnlichkeitsschwellwerten. Die Kosinusähnlichkeit dient hier als Referenzmaß. Die Schwellwerte wurden gleichmäßig aus dem Intervall $[0, 1]$ gewählt. Je Suchanfrage wurde also für alle Ähnlichkeitsschwellwerte je ein Wert für Precision und Recall ermittelt. Werden alle indizierten Dokumente, oder eine hinreichend große Stichprobe, je einmal als Suchdokument verwendet, erlauben die gemittelten Precision- und Recall-Werte bezüglich der Ähnlichkeitsschwellwerte eine Abschätzung der durchschnittlichen Werte von Precision und Recall einer beliebigen Suchanfrage.

Dieser Vergleichsansatz zur Untersuchung von Hashfunktionen folgt dem von Stein (2005a). Die präzise, gezielte Einstellung von Precision oder Recall der Hashfunktionen zur Ermittlung des jeweils anderen Wertes ist bislang nicht möglich, weshalb die Precision nicht direkt in Abhängigkeit vom Recall ermittelt werden konnte. Daher dient als Ausgangspunkt aller Experimente eine Einstellung der Hashfunktionen derart, dass die durchschnittliche Anzahl gefundener Dokumente einer Suchanfrage bei allen gleich groß ist. Damit ist keine Hashfunktion gegenüber anderen im Vorteil, da nicht durch eine höhere oder niedrigere Anzahl durchschnittlich gefundener Dokumente ein unverhältnismäßig höherer Recall bzw. eine höhere Precision erzielt werden kann. Im direkten Vergleich kann eine Funktion also nur dann besser als eine andere für einen bestimmten Ähnlichkeitsschwellwert sein, wenn im Verhältnis mehr relevante Dokumente gefunden werden. Die Einstellungen wurden für jeden Korpus neu bestimmt und waren auf allen aus ihm generierten Testkollektionen gleich.

4.3.1 Analyse der Testkollektionen

Zur besseren Einschätzung der Testergebnisse ist ein genauerer Blick auf die Testkollektionen notwendig. Untersucht wurde die Ähnlichkeitsverteilung der Dokumente in den jeweils kleinsten Testkollektionen, so dass für eine Reihe von Ähnlichkeitsschwellwerten die durchschnittliche Anzahl der Dokumente, die im gesamten Korpus mindestens diese Ähnlichkeit untereinander haben, eingeschätzt werden kann. Abbildung 4.4 zeigt die Ergebnisse.

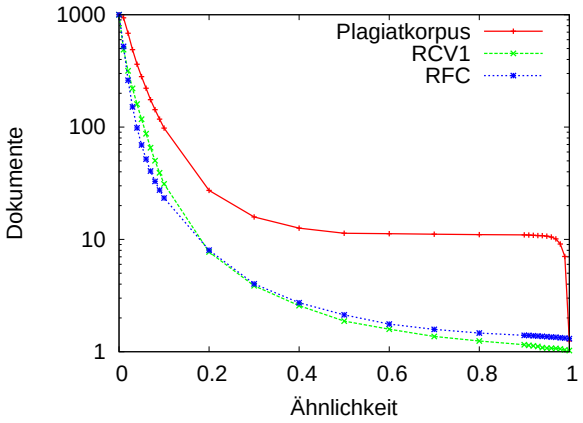

Abbildung 4.4: Die Grafik zeigt die Anzahl der Dokumente, die untereinander mindestens eine bestimmte Ähnlichkeit aufweisen. Verglichen werden Testkollektionen einer Größe von 1.000 Dokumenten aus dem Plagiatkorpus, dem RCV1 und der RFC-Kollektion.

Die Korpora RCV1 und RFC haben die Eigenschaft, dass im Durchschnitt sehr wenige Dokumente eine große Ähnlichkeit untereinander aufweisen. Die Hashfunktionen unterscheiden sich daher auf diesen Korpora nicht so stark im Recall. „Bessere“ Hashfunktionen können sich hier hauptsächlich durch eine höhere durchschnittliche Precision auszeichnen. Umgekehrt ist beim Plagiatkorpus aufgrund seiner Konstruktion eine verhältnismäßig große Anzahl von Dokumenten vorhanden, die sehr ähnlich zueinander sind. Dieser Korpus ist daher insbesondere für den Vergleich des Recalls der Hashfunktionen geeignet.

4.3.2 Vergleich der Hashfunktionen

In den Experimenten wurden die Hashfunktionen so konfiguriert, dass die durchschnittliche Anzahl gefundener Dokumente pro Suchanfrage 10 Dokumente betrug. Für Locality-Sensitive-Hashing ergaben sich zu diesem Zweck durchschnittlich $l = 17$ Instanzen der α-stabilen Hashfunktion. Für jede Instanz wurde der Parameter $k = 20$ eingestellt. Die Hashfunktionen für Fuzzy-Fingerprinting benötigten im Durchschnitt $|\Sigma| = 2$ Fuzzifizierungsschemata zu je $|\mu| = 3$ Intervallen.

Die Experimente zeigen, dass Fuzzy-Fingerprinting Locality-Sensitive-Hashing bei der Indizierung von Dokumenten deutlich überlegen ist. Auf allen Testkollektionen wurde ein größerer durchschnittlicher Recall und eine größere durchschnittliche Precision festgestellt. Abbildung 4.5 illustriert dies. Der signifikant bessere Recall beider Vertreter von Fuzzy-Fingerprinting belegt, dass die Einbeziehung von Domänenwissen sich vorteilhaft auf das Similarity-Hashing auswirkt. Locality-Sensitive-Hashing erreicht dagegen auf Grundlage der Randomisierung nur geringe Recall-Werte. Diese Ergebnisse sind unter dem Vorbehalt zu betrachten, dass die Dokumente im Plagiatkorpus künstlich generiert wurden. Das heißt, dass der Unterschied im Recall

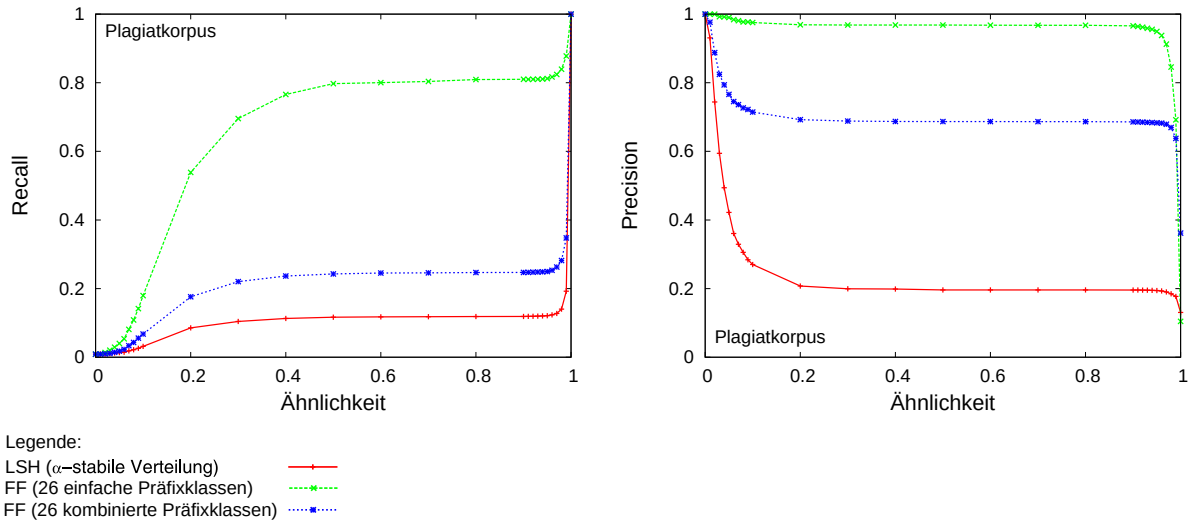

Abbildung 4.5: Die linke Grafik zeigt die Gegenüberstellung des Recalls von Hashfunktionen aus Locality-Sensitive-Hashing und Fuzzy-Fingerprinting für bestimmte Ähnlichkeitsschwellwerte, die rechte zeigt analog die Gegenüberstellung ihrer Precision. Es wurden Hashfunktionen basierend auf α-stabilen Wahrscheinlichkeitsverteilungen als Vertreter für LSH und Hashfunktionen basierend auf 26 Präfixklassen mit Präfixlänge 1 bzw. 26 kombinierten Präfixklassen als Vertreter für FF verwendet. Die Werte wurden auf Grundlage von Testkollektionen aus dem Plagiatkorpus der Größen 1.000, 2.000 und 3.000 Dokumente ermittelt und anschließend gemittelt.

und in der Precision nur das relative Verhältnis der Hashfunktionen zueinander zeigt. Es ist an dieser Stelle keine Aussage darüber möglich, ob Fuzzy-Fingerprinting oder Locality-Sensitive-Hashing auf realen Dokumenten überhaupt erfolgreich sind.

Demgegenüber zeigen die Experimente auf den übrigen Korpora, dass auch bei einer kleinen Anzahl von Dokumenten mit großer Ähnlichkeit noch Verbesserungen der Retrieval-Eigenschaften mit Fuzzy-Fingerprinting erzielt werden. Siehe dazu Abbildung 4.6. Die Verbesserung des Recalls ist hier nicht so stark ausgeprägt, die Precision verbessert sich jedoch deutlich.

Zur Beantwortung der Frage, wie stark die durchschnittliche Anzahl gefundener Dokumente mit der Anzahl indizierter Dokumente wächst, wurden die für die Experimente erstellen Indizes dahingehend ausgewertet. Am interessantesten ist diesbezüglich das Ergebnis auf Grundlage des RCV1, da hier die Anzahl indizierter Dokumente um mehrere Größenordnungen auf insgesamt 100.000 Dokumente gesteigert wurde. Abbildung 4.7 zeigt, dass die Anzahl gefundener Dokumente linear mit der Größe der Testkollektion wächst, wenn die Parameter der Hashfunktionen unangetastet bleiben.

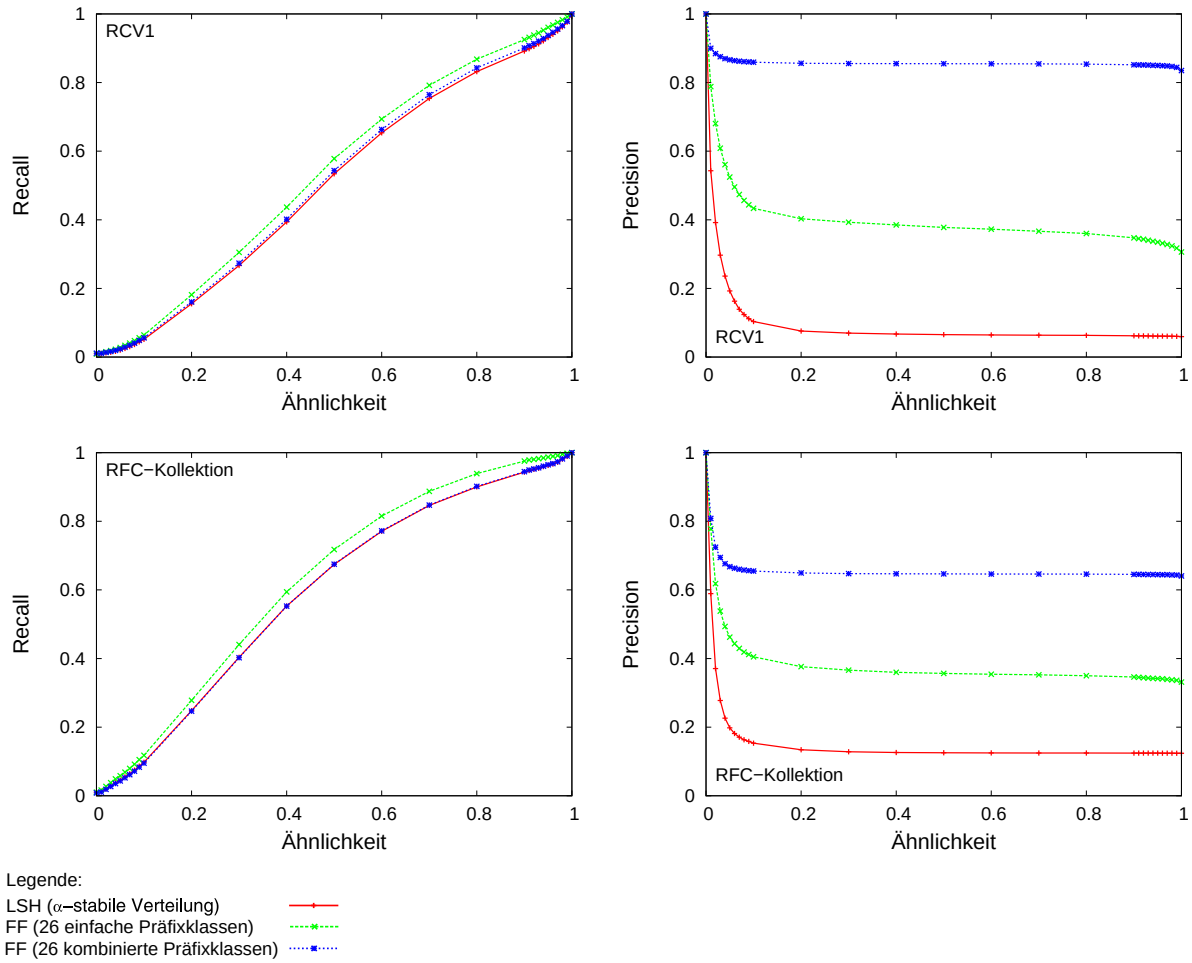

Abbildung 4.6: Die linken Grafiken zeigen die Gegenüberstellung des Recalls von Hashfunktionen aus Locality-Sensitive-Hashing und Fuzzy-Fingerprinting für bestimmte Ähnlichkeitsschwellwerte, die rechten zeigen analog die Gegenüberstellung ihrer Precision. Es wurden Hashfunktionen basierend auf α-stabilen Wahrscheinlichkeitsverteilungen als Vertreter für LSH und Hashfunktionen basierend auf 26 Präfixklassen mit Präfixlänge 1 bzw. 26 kombinierten Präfixklassen als Vertreter für FF verwendet. Die oberen Grafiken wurden auf Grundlage von Testkollektionen aus dem RCV1 der Größen 1.000, 10.000 und 100.000 Dokumente ermittelt und anschließend gemittelt. Die unteren Grafiken basieren analog auf Testkollektionen aus der RFC-Kollektion der Größen 1.000, 2.000 und 4.000 Dokumente.

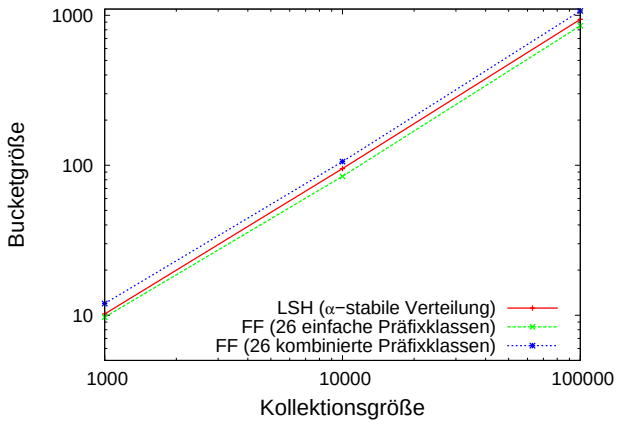

Abbildung 4.7: Die Grafik stellt die durchschnittliche Größe eines Buckets, das bei einer beliebigen Suchanfrage gefunden wird, der Größe der indizierten Dokumentkollektion gegenüber. Sie basiert auf den Testkollektionen aus dem RCV1 der Größen 1.000, 10.000 und 100.000 Dokumente.

Kapitel 5

Zusammenfassung und Ausblick

Die vorliegende Diplomarbeit definiert den Begriff des „Similarity-Hashing“ als gemeinsamen Nenner der beiden bisher bekannten Ansätze, Locality-Sensitive-Hashing und Fuzzy-Fingerprinting. Similarity-Hashing beschreibt den für Locality-Sensitive-Hashing konzipierten, allgemeinen Aufbau eines Indizierungsverfahrens für hochdimensionale Daten, nämlich eine Hashtabelle, deren Hashfunktion genau dann denselben Hashwert für zwei Dokumente berechnet, wenn sie ähnlich sind. Ähnliche Dokumente werden somit in der Hashtabelle im gleichen Bucket gespeichert. Die Suchzeit nach allen ähnlichen Dokumenten zu einem gegebenen Dokument verkürzt sich so auf die konstante Zeit, die für den Zugriff auf eine Hashtabelle benötigt wird.

Dieses Indizierungsverfahren wird in den Kontext anderer Verfahren zur Indizierung eingebettet, so dass eine Aufschlüsselung zwischen ihnen nach der Art der beantwortbaren Fragen und der dazu verwendeten Art der Indizierung möglich wird. Als Arten der Indizierung werden die beiden Prinzipien Inversion und Transformation identifiziert. Inverse Indizierung bedeutet die Umkehrung der Beziehung zwischen indizierten Objekten und Frageobjekten, also zum Beispiel zwischen Dokumenten und Worten. Im Gegensatz dazu wird bei transformativer Indizierung sowohl das indizierte Objekt als auch das Frageobjekt mithilfe einer Transformationsvorschrift in einen gemeinsamen Raum eingebettet. Im Vergleich zu den erfolgreichsten Vertretern beider Arten der Indizierung für die jeweiligen Fragetypen zeigt sich, dass Similarity-Hashing das beste Indizierungsverfahren zur Beantwortung von Fragen nach Objekten aus hochdimensionalen Räumen ist. Darüber hinaus wird die Verwandtschaft von Similarity-Hashing zum einzigen anderen bekannten transformativen Indizierungsverfahren, der Signaturliste, deutlich.

Die Konstruktionsprinzipien von Hashfunktionen für das Similarity-Hashing werden aufgearbeitet. Es stellt sich heraus, dass die vier grundlegenden Schritte Quantifizierung von Dokumenten, Dimensionsreduktion der Dokumentvektoren, Abbildung der Dimensionen auf natürliche

Zahlen und ihre Kombination zu einem Hashwert unterschieden werden können.

Locality-Sensitive-Hashing und Fuzzy-Fingerprinting werden im Detail erläutert. Locality-Sensitive-Hashing ist ein Rahmenwerk, in dem Hashfunktionen randomisiert konstruiert werden. Die Bestandteile der konstruierten Hashfunktionen werden zufällig aus einer Familie einfacher Hashfunktionen gewählt. Die vorgestellten Familien basieren auf der Einbettung zu hashender Objekte in den Hamming-Space bzw. auf α-stabilen Verteilungen. Für sie kann die Wahrscheinlichkeit, mit der die Hashwerte zweier Dokumente einer beliebigen Funktion aus der Familie für beide gleich sind, abhängig von ihrer Ähnlichkeit berechnet werden. Darauf aufbauend erlaubt das Rahmenwerk die Konstruktion von Hashfunktionen, mit denen im Similarity-Hashing untere Schranken für die Wahrscheinlichkeit der Kollision der Hashwerte ähnlicher Dokumente, abhängig von einem Ähnlichkeitsschwellwert, garantiert werden können.

Fuzzy-Fingerprinting basiert dagegen auf der Ausnutzung domänenspezifischen Wissens zur Konstruktion von Hashfunktionen. Die betrachtete Hashfunktion errechnet Hashwerte für Dokumente auf Grundlage eines niedrigdimensionalen Dokumentmodells, dem Präfixklassenmodell. Das Konzept von Fuzzy-Fingerprinting wird generalisiert zu einem Rahmenwerk für kontextsensitive Hashfunktionen. Die Konstruktion kontextsensitiver Hashfunktionen für Fuzzy-Fingerprinting basiert auf der Auswahl einer kleinen Anzahl von Charakteristiken, für die ein Erwartungswert bekannt ist. Durch Normalisierung der Werte der Charakteristika einer konkreten Objektinstanz mit ihren jeweiligen Erwartungswerten wird ihre Diskriminanzkraft vereinheitlicht.

Zu der Fragestellung, welcher der beiden Ansätze für den konkreten Anwendungsfall der Indizierung von Dokumenten besser geeignet ist, wurden breit angelegte Experimente durchgeführt. Die Analyse erfolgte anhand des Vergleichs von Recall und Precision beider Ansätze für eine Reihe von Testkollektionen aus verschiedenen Dokumentkorpora. Es hat sich gezeigt, dass die Ausnutzung von Domänenwissen beim Fuzzy-Fingerprinting dem allgemeinen Ansatz von Locality-Sensitive-Hashing überlegen ist. In allen Experimenten waren die Recall- und Precision-Werte von Fuzzy-Fingerprinting größer als die von Locality-Sensitive-Hashing. Das betrifft sowohl die Korpora mit künstlich konstruierten Dokumenten, in dem besonders viele Dokumente mit großen Ähnlichkeiten zueinander enthalten sind, als auch die Korpora, die aus realen Dokumenten bestehen. Auf algorithmischer Ebene wurde gezeigt, dass beide Ansätze unter denselben Grundvoraussetzungen gleich gut skalieren. Mit steigender Anzahl zu indizierender Dokumente steigt die durchschnittliche Anzahl gefundener Dokumente pro Suche linear. Das Verhältnis der Kollektionsgröße zur Anzahl gefundener Dokumente pro Suche bleibt also erhalten.

Zur Klärung der Frage, wie sich das zur Quantifizierung von Dokumenten für Fuzzy-Fingerprinting vorgeschlagene Präfixklassenmodell gegenüber dem Vektorraummodell verhält, wurden Experimente in Form von Clusteranalysen durchgeführt und das Qualitätsmaß Expected-Density verwendet. Beide Testverfahren haben gezeigt, dass das Präfixklassenmodell Dokumente

schlechter repräsentiert. Grund dafür ist die stärkere Abstraktion im Verhältnis zum Vektorraummodell. Ein Verbesserungsvorschlag für das Modell wurde gemacht, der vorsieht, dass die Präfixklassen abhängig von ihren jeweiligen Erwartungswerten kombiniert werden, so dass sich für jede Kombination eine gleich hohe Summe der einzelnen Erwartungswerte ergibt. Die Experimente haben gezeigt, dass, auch bei Verwendung weniger Präfixklassenkombinationen, Dokumente besser repräsentiert werden als mit dem ursprünglich vorgeschlagenen Modell. Wird dieser Ansatz beim Fuzzy-Fingerprinting eingesetzt, ist die Einstellung der Genauigkeit der Hashwerte möglich. Das wiederum ist ein zusätzlicher Ansatz zur Beeinflussung von Precision und Recall der Hashfunktion.

Similarity-Hashing ist ein sehr spannendes Forschungsgebiet. Die Suche in hochdimensionalen Räumen spielt in vielen Problemstellungen eine Hauptrolle, sei es im textbasierten Information-Retrieval oder anderen Domänen. Oftmals hängt die Laufzeit der eingesetzten Lösungsstrategien allein von der Laufzeit der Suche ab. Similarity-Hashing ist das erste Verfahren, das die Suchgeschwindigkeit in hochdimensionalen Daten gegenüber der einzig sinnvollen Alternative einer sequentiellen Suche verbessert, ohne dass sich die Suchergebnisse dabei zu stark verschlechtern. Die in dieser Arbeit vorgenommene Einordnung von Similarity-Hashing in den Kontext der Indizierung und die Gegenüberstellung aller bisher bekannten Ansätze vermitteln ein grundlegendes Verständnis, so dass es in verschiedenen Problemstellungen eingesetzt werden kann.

Es verbleiben noch eine Reihe offener Fragen für zukünftige Forschungen. Im Speziellen kann die Überlegenheit von Fuzzy-Fingerprinting bei der Indizierung von Dokumenten bislang nicht formal belegt werden. Die Frage, die sich daraus ergibt, ist, ob hier analog zu Locality-Sensitive-Hashing eine untere Schranke für die Wahrscheinlichkeit der Kollision der Hashwerte ähnlicher Dokumente gezeigt werden kann. Darüber hinaus ist für beide Verfahren der Zusammenhang zwischen den Zielgrößen ε, der Kollisionswahrscheinlichkeit und der durchschnittlichen Anzahl gefundener Dokumente pro Suche unbekannt. Außerdem ist eine weitere Analyse des Objektraums interessant, in den die Hashfunktionen Dokumente abbilden. Interessant dabei ist wie sich verschiedene Hashwerte aus dem Objektraum zueinander verhalten, da die Ähnlichkeit zweier Hashwerte, wenn sie zum Beispiel nur wenige unterschiedliche Ziffern aufweisen, Rückschlüsse auf die Ähnlichkeit der Dokumente zulässt, die in dem durch sie jeweils bezeichneten Bucket gespeichert werden.

Anhang A

Clusteranalyse

Ein Clustering $\mathcal{C} = \{C_1, \ldots, C_K\}$ für eine Dokumentmenge $\mathbf{D}$ ist eine Menge von K Clustern, die jeweils Teilmengen von $\mathbf{D}$ sind. Tan u. a. (2005) unterscheiden sechs Typen von Clusterings, die sich in drei miteinander kombinierbare Gruppen unterteilen. Abbildung A.1 illustriert die verschiedenen Typen.

(1.) Hierarchische Clusterings unterscheiden sich von partitionierten darin, dass bei ihnen Cluster in einer Baumstruktur angeordnet sind. Der Inhalt eines Clusters C_i ist in diesem Fall die Vereinigung einer Menge anderer Cluster, die als Nachfolger von C_i und als feinere Unterteilung aufzufassen sind. Insbesondere gibt es einen Cluster, der $\mathbf{D}$ entspricht und keinen, der Teil mehrerer anderer ist. Partitionierende Clusterings sind dagegen Unterteilungen der Dokumente, so dass jedes Dokument genau einem Cluster zugeordnet ist. (2.) Eine exklusive Zuordnung der Dokumente zu genau einem Cluster bzw. genau einem Pfad in der Hierarchie wird nicht immer vorausgesetzt, so dass auch Überlappungen von Clustern zulässig sind. Überlappende Clusterings sind also Aufteilungen, bei denen ein Dokument mindestens einem Cluster zugeordnet ist. Bei nicht überlappenden Clusterings gilt, dass höchstens ein Cluster ein bestimmtes Dokument enthält. (3.) Wenn für ein Clustering $\mathcal{C}$ gilt, dass $\bigcup_{C_i \in \mathcal{C}} C_i = \mathbf{D}$ ist, dann handelt es sich um ein vollständiges Clustering, wohingegen andernfalls ein unvollständiges oder partielles Clustering vorliegt.

A.1 Clusteralgorithmen

Die Clusteranalyse zur Bewertung von Dokumentmodellen geschieht im Allgemeinen auf der Grundlage eines vollständigen, nicht überlappenden, partitionierten Clusterings. Solche Clusterings werden mithilfe von Clusteralgorithmen jeweils für alle Dokumentmodelle generiert. Dies geschieht unüberwacht, also ohne externes Wissen oder Hilfe. Die bekannten Algorithmen

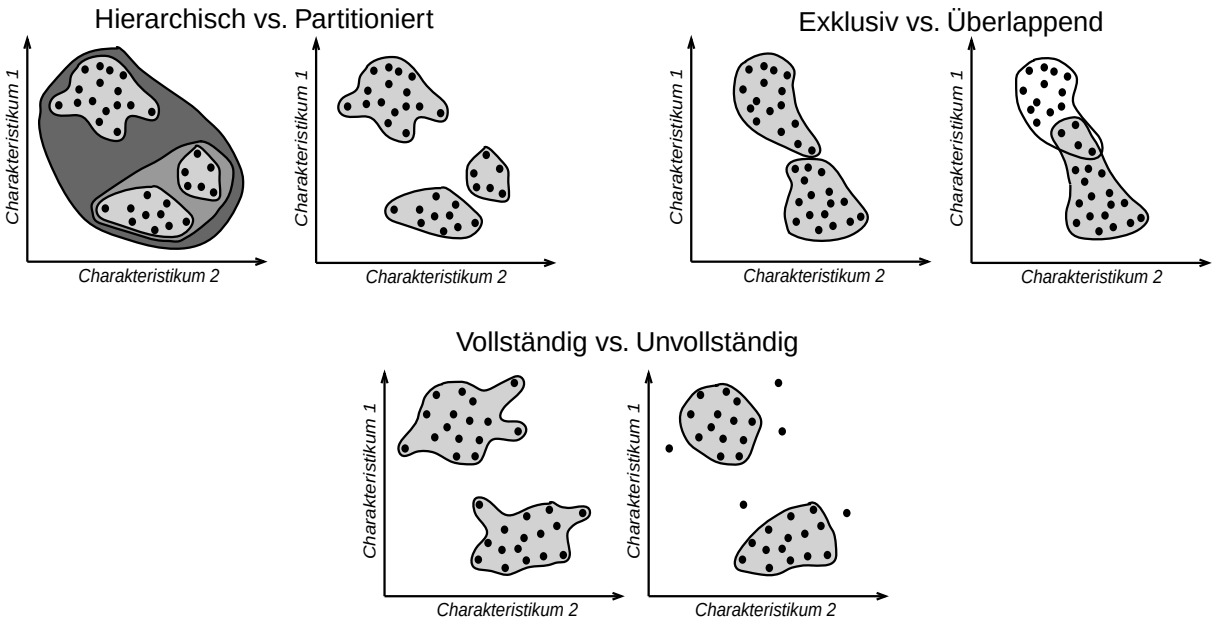

Abbildung A.1: Die Abbildung zeigt die drei Gruppen von Clusterings, in denen sich je zwei Clustering-Typen gegenüberstehen. Jeder Typ einer Gruppe ist mit dem einer anderen kombinierbar.

lassen sich nach Stein (2005b) in verschiedene Kategorien einordnen, die in Abbildung A.2 zu sehen sind.

Hierarchische Algorithmen erzeugen inkrementell ein hierarchisches Clustering. Ausgehend von einem (trivialen) Initialclustering $\mathcal{C}_0$ werden Cluster aufgrund von Auswahlheuristiken vereinigt oder geteilt, so dass nach n Schritten das Clustering $\mathcal{C}_n$ maximal vereinigt bzw. geteilt ist. Die agglomerative Variante initiiert n Cluster, einen für jedes Dokument, wohingegen die divisive mit genau einem Cluster beginnt, der alle Dokumente enthält. Um ein partitioniertes Clustering zu erhalten, wird der Algorithmus nach dem i-ten Schritt abgebrochen und das aktuelle $\mathcal{C}_i$ als Ergebnis betrachtet. Die Schwierigkeit hierarchischer Algorithmen besteht einerseits in der Konstruktion geeigneter Auswahlheuristiken zur Auswahl zu teilender bzw. zu vereinender Cluster und andererseits in der Definition eines (optimalen) Abbruchkriteriums. Als nachteilig

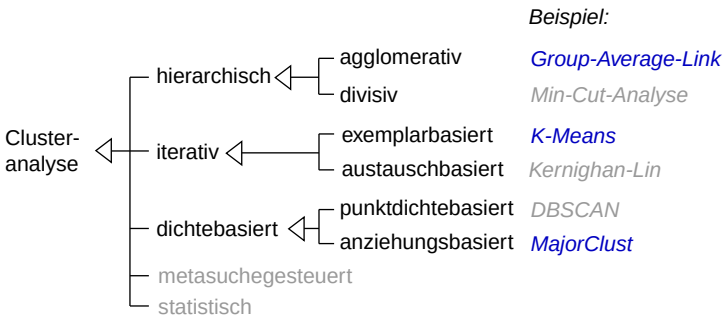

Abbildung A.2: Die Taxonomie nach Stein (2005b) zeigt eine Unterteilung von Clusteralgorithmen in verschiedene Kategorien.

erweist sich die inkrementelle Natur der Algorithmen, denn eine einmal getroffene Entscheidung, einen Cluster zu teilen bzw. zwei zu vereinen, kann nicht wieder rückgängig gemacht werden, auch dann nicht, wenn sie sich später als nachteilig herausstellt.

Iterative Algorithmen zur Clusteranalyse basieren auf der wiederholten Ausführung derselben Schritte, bis ein Optimalitätskriterium (hinreichend) erfüllt ist. Dabei greifen sie auf die Ergebnisse der vorigen Iteration zurück, um das Clustering Schritt für Schritt zu optimieren. Ein Initialclustering $\mathcal{C}_0$, bestehend aus K zufällig oder mithilfe einer Auswahlheuristik gewählten Clustern, dient als Ausgangspunkt der Algorithmen. Austauschbasierte Algorithmen wählen mindestens zwei Dokumente aus verschiedenen Clustern und vertauschen sie. Hat sich das Clustering dadurch verbessert, wird das Ergebnis in der nächsten Iteration beibehalten. Exemplarbasierte Algorithmen erzeugen in jeder Iteration ein von Grund auf neues Clustering. Als Ausgangspunkt dafür dienen K repräsentative — also exemplarische — Punkte. Für jeden Cluster des vorherigen Clusterings wird ein solcher Punkt ermittelt. Anschließend werden die Dokumente neuerlich verteilt, wobei ein Dokument dem i-ten Cluster zugeordnet wird, falls es dem i-ten Repräsentanten am nächsten liegt. Den iterativen Verfahren ist gemein, dass sie über die Anzahl K der in $\mathbf{D}$ vorhandenen Cluster informiert werden müssen. Sind Informationen darüber nicht verfügbar, bleibt nur die Möglichkeit sie mit verschiedenen Werten für K auszuführen und anschließend unter den Ergebnissen das beste zu wählen. Im Fall der Bewertung von Dokumentmodellen ist K bekannt.

Dichtebasierte Algorithmen unterteilen sich in den punktdichtebasierten DBSCAN und den anziehungsbasierten MajorClust. Bei beiden hängt die Zuordnung eines Dokuments $\mathbf{d}$ zu einem bestimmten Cluster davon ab, ob die zu $\mathbf{d}$ ähnlichen Dokumente ebenfalls diesem Cluster zugeordnet wurden. Der punktdichtebasierte Ansatz ermittelt für jeden Punkt die Dichte der Punkte in seiner unmittelbaren Umgebung. Abhängig davon, wie groß der Dichtewert eines Punktes ist, werden drei Typen unterschieden, nämlich Zentral-, Rand- und Rauschpunkte. Jeder Zentralpunkt wird mit den in seiner unmittelbaren Umgebung liegenden Zentral- und Randpunkten zu einem Cluster zusammengefasst. Alle Rauschpunkte werden in einem gemeinsamen Cluster vereint. Der anziehungsbasierte Ansatz dagegen interpretiert die Ähnlichkeit zwischen zwei Dokumenten als Gravitation, so dass ein Dokument genau dem Cluster zugeordnet wird, von dem die größte Anziehungskraft ausgeht. Berücksichtigt wird nur die Gravitation hinreichend ähnlicher Dokumente. Die Parameter beider Verfahren sind Ähnlichkeitsschwellwerte, die definieren, ab welcher Ähnlichkeit ein Dokument in die Dichte- bzw. Gravitationsberechnung für ein gegebenes Dokument einfließt. Bei DBSCAN sind zusätzlich noch Dichteintervalle zu definieren, die die Zuordnung von Dichtewerten für Dokumente auf die drei Punkttypen erlauben.

Von der Erläuterung der metasuchegesteuerten und der statistischen Verfahren wird an dieser Stelle abgesehen, da hier der vorgegebene Rahmen üblicher Clusteralgorithmen zur Bewertung von Dokumentmodellen im Information-Retrieval verlassen würde.

Abhängig von der Art der zugrundeliegenden Struktur der Dokumente differiert die Erkennungsleistung der drei vorgestellten Klassen von Algorithmen. Interessant sind dabei insbesondere die Fälle, in denen die Algorithmen nicht das erwartete Ergebnis produzieren, also eine schlechte Unterteilung generieren. Der bedeutendste Fall beim Clustern von Dokumenten ist das Auftreten verschränkter Cluster. Die Cluster haben dabei meist eine konkave Form und umschließen sich gegenseitig teilweise oder vollständig. Es ist anzunehmen, dass in den hochdimensionalen Vektorräumen der Dokumentmodelle solche Muster häufig auftreten. Einige Varianten hierarchischer und iterativer Algorithmen haben Probleme diese Cluster zu erkennen. Dichtebasierte Algorithmen hingegen werden weniger stark davon beeinflusst.

Insgesamt lässt sich kein eindeutiges Gesamturteil fällen, welcher der Ansätze den anderen überlegen ist. Abhängig von der Beschaffenheit der Dokumente, den gewählten Parametern und gegebenenfalls der zufälligen Initialisierung, kann jeder Algorithmus die anderen übertreffen. Insofern ist es nicht sinnvoll, die Dokumente anhand eines bestimmten Algorithmus' zu bewerten, sondern vielmehr sollten die Ergebnisse verschiedener Algorithmen gemittelt werden. Im Folgenden werden die drei für die Bewertung gewählten Algorithmen näher vorgestellt.

A.1.1 Group-Average-Link

Bei Group-Average-Link handelt es sich um einen hierarchisch-agglomerativen Clusteralgorithmus. Die Auswahl der beiden Cluster C_1 und C_2, die im i-ten Schritt miteinander vereint werden, geschieht mithilfe einer Distanzfunktion $\sigma_{\mathcal{C}}(C_1, C_2)$. Ist die Distanz zwischen C_1 und C_2 am geringsten unter allen möglichen Paarungen, werden sie vereint. Die in Group-Average-Link eingesetzte Distanzfunktion ist nachfolgend aufgeführt und berechnet die gemittelten paarweisen Ähnlichkeiten der Dokumente aus C_1 und C_2.

$$\sigma_{\mathcal{C}}(C_1, C_2) = \frac{1}{|C_1| \cdot |C_2|} \sum_{d_i \in C_1} \sum_{d_j \in C_2} \sigma(d_i, d_j)$$

Der Unterschied zwischen Group-Average-Link und anderen hierarchisch-agglomerativen Clusteralgorithmen besteht einzig in der Berechnung von $\sigma_{\mathcal{C}}$. Andere Funktionen berücksichtigen beispielsweise nur die nächsten oder am weitesten entfernten Nachbarn beider Cluster (Single- bzw. Complete-Link).

Werden die Cluster C_1 und C_2 vereinigt, ist die Neuberechnung der Distanzen von $C_1 \cup C_2$ zu allen übrigen Clustern $C_i \in \mathcal{C} \backslash \{C_1, C_2\}$ für den nächsten Schritt notwendig. Damit hierfür kein quadratischer Aufwand für die Distanzberechnungen entsteht, geschieht die Neuberechnung mithilfe der aus dem vorigen Schritt bekannten Distanzen $\sigma_{\mathcal{C}}(C_1, C_i)$ und $\sigma_{\mathcal{C}}(C_2, C_i)$ nach der

Lance-Williams-Formel[1]:

$$\sigma_{\mathcal{C}}(C_1 \cup C_2, C_i) = \frac{|C_1|}{|C_1 \cup C_2|} \cdot \sigma_{\mathcal{C}}(C_1, C_i) + \frac{|C_2|}{|C_1 \cup C_2|} \cdot \sigma_{\mathcal{C}}(C_2, C_i)$$

Die in den Experimenten verwendete Variante von Group-Average-Link wurde um ein Abbruchkriterium erweitert, damit ein partitioniertes Clustering erzeugt wird. Analog zu iterativen Algorithmen handelt es sich um die Anzahl K erwünschter Cluster. Group-Average-Link vereinigt hier also genau $n - K$ mal zwei Cluster.

A.1.2 K-Means

Unter den iterativ-exemplarbasierten Clusteralgorithmen ist K-Means der bekannteste. Sein Name ist wörtlich zu verstehen, denn mit K-Means werden K Mittelwerte (engl. Means) errechnet. Für K exemplarische Punkte $\mathbf{c}_1, \ldots, \mathbf{c}_K$, so genannte Zentroide, die je einen Cluster repräsentieren, werden die folgenden zwei Schritte wiederholt ausgeführt: (1.) Jedes Dokument $\mathbf{d} \in \mathbf{D}$ wird dem i-ten Cluster C_i zugeordnet, falls $i = \arg\min_{j \in \{1,\ldots,K\}} \sigma_{\mathbf{c}}(\mathbf{d}, \mathbf{c}_j)$ ist. (2.) Jeder Zentroid wird abhängig von den seinem Cluster zugeordneten Dokumenten und einem Optimierungskriterium neu errechnet. Beide Schritte werden wiederholt, bis sich alle Zentroide stabilisiert haben, so dass ihre Position in zwei aufeinanderfolgenden Durchläufen hinreichend gleich bleibt.

Die Distanz $\sigma_{\mathbf{c}}$ zwischen einem Dokument $\mathbf{d}$ und einem Zentroid kann zum Beispiel mithilfe der Kosinusähnlichkeit errechnet werden, obwohl die Zentroide selbst keine Dokumente repräsentieren. Die Neuberechnung des i-ten $\mathbf{c}_i$ geschieht durch komponentenweise Bestimmung der Mittelwerte der in C_i enthaltenen Dokumentmodelle. Dieses Vorgehen zur Neuberechnung der Zentroide minimiert die Summe der quadratischen Abweichungen aller Dokumentmodelle von $\mathbf{c}_i$, was gleichzeitig das Optimierungskriterium ist:

$$e(\mathcal{C}) = \sum_{j=1}^{K} \sum_{\mathbf{d} \in C_j} (\mathbf{c}_j - \mathbf{d})^2$$

Durch Ableitung von $e(\mathcal{C})$ nach $\mathbf{c}_i$ und Gleichsetzung mit 0 ist zu sehen, dass die komponentenweise Mittelwertbestimmung $e(\mathcal{C})$ für $\mathbf{c}_i$ minimiert. Der Algorithmus verbessert folglich mit jeder Iteration die Zentroide, rückt sie also dem jeweiligen Zentrum der K Cluster näher. Das Resultat entspricht jedoch nicht zwangsläufig der optimalen Unterteilung. Es hängt viel von der Anfangsposition der Zentroide ab. Bei einer ungünstigen Verteilung über den Vektorraum kann

[1] Die gezeigte Formel ist auf Group-Average-Link spezialisiert; sie wurde abgeleitet von einer allgemeinen Formel für alle Varianten hierarchisch-agglomerativer Algorithmen.

der Algorithmus schlechte Resultate liefern. Das Gleiche gilt bei ineinander verschachtelten Clustern oder Clustern mit extremen Größendifferenzen.

Die einfachste Art K Zentroide zu initialisieren ist, zufällig K Dokumente zu wählen, deren Dokumentmodelle als Startpunkt der Zentroide dienen. Dies macht aber auch Fälle wahrscheinlich, in denen mehrere Zentroide gehäuft auftreten oder Dokumente weit entfernt von allen übrigen gewählt werden. Darüber hinaus gibt es noch zwei andere Möglichkeiten. Die erste besteht darin, eine zufällige, kleine Auswahl der Dokumente losgelöst zu clustern. Für die sich daraus ergebenden Cluster werden die Zentroide berechnet und anschließend alle Dokumente geclustert. Die zweite verwendet zu Beginn zufällig gewählte Zentroide und verändert die Position jedes einzelnen anschließend so, dass er am weitesten von allen anderen weg liegt.

A.1.3 MajorClust

MajorClust nach Stein und Niggemann (1999) ist ein dichtebasierter Clusteralgorithmus, dessen Idee auf der Ausnutzung der intrinsischen Ähnlichkeitsbeziehungen der Dokumente untereinander basiert. Dafür wird das Prinzip „Gravitation“ operationalisiert, so dass ein Dokument durch den Algorithmus genau dem Cluster zugeordnet wird, von dem es am stärksten angezogen wird. Jedes Dokument wird dabei von jedem anderen nur so stark angezogen, wie es ihm ähnlich ist. Die Gravitation, die ein Cluster auf ein Dokument ausübt, entspricht der kumulierten Gravitation der enthaltenen Dokumente.

Die algorithmische Grundlage von MajorClust ist der ungerichtete, gewichtete ε-Ähnlichkeitsgraph $G = \langle V, E, \varepsilon \rangle$, wobei $V = \mathbf{D}$ gilt und eine Kante zwischen zwei Dokumenten $\mathbf{d}_1$ und $\mathbf{d}_2$ nur dann existiert, wenn ihre Ähnlichkeit $\varphi(\mathbf{d}_1, \mathbf{d}_2) \geq \varepsilon$ ist. ε ist aus dem Intervall $[0, 1]$. Jede Kante ist mit der Ähnlichkeit der verbunden Dokumente gewichtet. Angenommen, $\mathcal{C} = \{C_1, \ldots, C_K\}$ ist ein Clustering der Dokumente, unter der Bedingung, dass jeder Cluster Dokumente enthält, die einen zusammenhängenden Teilgraphen mit Rücksicht auf ε induzieren. Die relative Güte von $\mathcal{C}$ ist dann definiert als *gewichteter partieller Kantenzusammenhang*:

$$\Lambda(\mathcal{C}) = \sum_{i=1}^{K} |C_i| \cdot \bar{\lambda}_i$$

Der Kantenzusammenhang λ eines Graphen ist die Anzahl der Kanten, ohne die der Graph nicht mehr zusammenhängend ist, was sich analog auch für beliebige Teilgraphen errechnen lässt. Um die Kantengewichte eines gewichteten Graphen hier zu berücksichtigen, wird $\bar{\lambda}$ verwendet; hierbei handelt es sich um die minimale Summe der Gewichte der Kanten, ohne die ein (Teil-)Graph nicht mehr zusammenhängend ist. Per Definition bedeutet ein höherer Kantenzusammenhang $\bar{\lambda}$ des durch einen Cluster induzierten Teilgraphen, dass, abhängig von ε, viele

Dokumente einander ähnlich sind und sich gewissermaßen gegenseitig „anziehen“. Ein Algorithmus, der ein Clustering $\mathcal{C}$ so erzeugt, dass Λ maximal wird, erkennt also automatisch eine optimale Unterteilung der Dokumente.

MajorClust ist ein heuristischer Algorithmus zur lokalen Optimierung von Λ: Als Initialclustering $\mathcal{C}_0$ wird jedes Dokument einem eigenen Cluster zugeordnet. In der Folge werden alle Dokumente in zufälliger Reihenfolge durchgegangen und ein Dokument dem Cluster zugeordnet, zu dem die gewichtete Mehrheit seiner im ε-Ähnlichkeitsgraphen benachbarten Dokumente gehört. Wenn mehrere Cluster zur Auswahl stehen, wird einer zufällig gewählt. Wechselt bei einem Durchlauf kein Dokument mehr den Cluster, terminiert der Algorithmus.

Meyer zu Eissen und Stein (2002) präsentieren die Ergebnisse einer umfangreichen Untersuchung und Gegenüberstellung hierarchischer, iterativer und dichtebasierter Algorithmen. Für K-Means und MajorClust ergeben sich, die eingesetzten Bewertungsmaße berücksichtigend, jeweils gute Analyseergebnisse. Der hierarchische Single-Link-Algorithmus schneidet hingegen schlecht ab.

A.2 Clustervalidierung

Die Bewertung eines Clusterings geschieht mithilfe eines Bewertungs- oder Validitätsmaßes. „Validität“ bedeutet in diesem Zusammenhang, dass gemessen wird, wie gut ein Clustering die tatsächliche Struktur der Dokumente widerspiegelt bzw. wie es sich relativ zu anderen Clusterings verhält. Je nach Art der Validitätsmessung werden externe Informationen benötigt, so dass externe und interne Validitätsmaße zu unterschieden sind. Einen Vergleich verschiedener Validitätsmaße geben Halkidi u. a. (2002) und Stein u. a. (2003).

Im Falle der Bewertung von Dokumentmodellen und ihrer Gegenüberstellung wird als Vergleichspunkt in den meisten Fällen ein Referenzclustering $\mathcal{C}^\star$ verwendet und ein externes Validitätsmaß eingesetzt. Das Referenzclustering basiert dabei auf der durch Dritte vorgenommenen Unterteilung der Dokumente in inhaltliche Kategorien, die als Referenzcluster betrachtet werden. Eine solche Unterteilung liegt beispielsweise im oben vorgestellten Dokumentkorpus RCV1 vor. Da die dortige Unterteilung manuell vorgenommen wurde, ist anzunehmen, dass sie der Optimalfall ist, den es mithilfe geeigneter Dokumentmodelle und Clusteralgorithmen zu erreichen gilt. Dabei wird außer Acht gelassen, dass dieses Vorgehen auch fehlerbehaftet sein kann, also Dokumente nach mehrheitlicher Meinung falsch einsortiert wurden.

Das F-Measure ist ein Validitätsmaß zur Bewertung eines Clusterings $\mathcal{C} = \{C_1, \ldots, C_K\}$ gegenüber einem Referenzclustering $\mathcal{C}^\star = \{C_1^\star, \ldots, C_l^\star\}$. Es basiert auf der Berechnung des harmonischen Mittels von Precision und Recall[2] für jede Paarung (i, j) der Cluster $C_i^\star \in \mathcal{C}^\star$ und

$C_j \in \mathcal{C}$:

$$F(i,j) = \frac{2 \cdot prec(i,j) \cdot rec(i,j)}{prec(i,j) + rec(i,j)}$$

Verallgemeinert berechnet sich der F-Measure-Wert für $\mathcal{C}$ und $\mathcal{C}^\star$ wie folgt:

$$F = \sum_{i=1}^{l} \frac{|C_i^\star|}{n} \cdot \max_{j=1,\ldots,K} \{F(i,j)\}$$

Als Basislinie für die Bewertung dient meistens entweder ein naives oder das am weitesten verbreitete Modell. Hier trifft letzteres zu: Das klassische Vektorraummodell wird als Basislinie verwendet. Es ist das am häufigsten verwendete Dokumentmodell im Information-Retrieval. Abhängig vom Abschneiden der zu testenden Modelle gegenüber dem Vektorraummodell wird sowohl eine Vergleichbarkeit ihres Vermögens, Dokumente zu repräsentieren, geschaffen, als auch ihre Praxistauglichkeit demonstriert.

A.2.1 Probleme bei der Bewertung von Clusterings

Wie oben schon erwähnt, ist es nicht sinnvoll, Dokumentmodelle nur auf Grundlage der Ergebnisse eines einzelnen Clusteralgorithmus zu vergleichen. Das würde die Aussage, ob ein Dokumentmodell ein Dokument besser repräsentieren kann als ein anderes, davon abhängig machen, ob der eingesetzte Clusteralgorithmus von seinen Besonderheiten profitiert. Die zahlreichen Parameter der Algorithmen erschweren die Bewertung zusätzlich. Es stellt sich die Frage, welche der Ergebnisse verschiedener Algorithmen zu berücksichtigen sind: Die besten, die schlechtesten oder der Durchschnitt?

Eine alternative Bewertungsmethode haben Stein u. a. (2003) vorgeschlagen. Sie schlagen ein Maß zur Bewertung der intrinsischen Ähnlichkeitsbeziehungen der Dokumente untereinander auf Basis ihres ε-Ähnlichkeitsgraphen vor. Das Referenzclustering als Schablone für den Graphen verwendend, wird gemessen, wie stark die Inter- und Intraclusterähnlichkeit durch ein bestimmtes Dokumentmodell das Referenzclustering widerspiegelt. Das dafür eingesetzte Maß errechnet die erwartete Dichte der Kanten und heißt demzufolge Expected-Density $\bar{\rho}$.

[2] Die Precision, wie sie in Abschnitt 1.7 definiert ist, der Anteil relevanter an den erhaltenen Dokumenten und der Recall der Anteil relevanter, erhaltener Dokumente an allen relevanten, hier also $prec(i,j) = |C_j \cap C_i^\star|/|C_j|$ und $rec(i,j) = |C_j \cap C_i^\star|/|C_i^\star|$.

A.3 Expected-Density

Ein Graph $G = \langle V, E \rangle$ wird *dünn* genannt, wenn die Anzahl der Kanten in ihm linear mit der Anzahl der Knoten wächst, so dass $|E| = \mathcal{O}(|V|)$ gilt. Im Extremfall wächst die Kantenmenge jedoch quadratisch mit der Knotenmenge, also $|E| = \mathcal{O}(|V|^2)$. Ein Graph, auf den das zutrifft, wird *dicht* genannt. Die Dichte θ eines Graphen kann also durch $|E| = |V|^\theta$ beziffert werden. Für einen gewichteten Graphen lässt sie sich genauso mithilfe von $w(G) = |V| + \sum_{e \in E} w(e)$ errechnen:

$$w(G) = |V|^\theta \qquad \Leftrightarrow \qquad \theta = \frac{\ln w(G)}{\ln |V|}$$

$w(G)$ ist die Summe aller Kantengewichte aus G, zu der V addiert wird, um Gewichte im Bereich $[0, 1]$ zu berücksichtigen. Im Falle eines Ähnlichkeitsgraphen, in dem e die Dokumente $\mathbf{d}_1$ und $\mathbf{d}_2$ verbindet, ist das Kantengewicht $w(e) = \varphi(\mathbf{d}_1, \mathbf{d}_2)$.

Angenommen, $\mathcal{C}^\star = \{C_1^\star, \ldots, C_l^\star\}$ ist ein Referenzclustering der Dokumente $\mathbf{D}$, für die G ein Ähnlichkeitsgraph ist. Der i-te Cluster $C_i^\star$ aus $\mathcal{C}^\star$ induziert einen Teilgraphen $G_i = \langle V_i, E_i \rangle$ in G. Wenn die Dichte in G_i im Vergleich zu G größer ist, dann ist die Summe der Kantengewichte zwischen allen Knoten aus G_i größer als erwartet. Die Knoten in G_i sind also verhältnismäßig stark miteinander verbunden. Daraus folgt, dass ein beliebiger Clusteralgorithmus mit größerer Wahrscheinlichkeit die Dokumente aus G_i einem Cluster zuordnet, der $C_i^\star$ gleicht. Das Dokumentmodell, mit dem die Kantengewichte in G berechnet wurden, ist also dazu geeignet, $C_i^\star$ gegenüber anderen Clustern erkennbar zu machen.

θ kann verwendet werden, um die Dichte von G_i gegenüber G zu ermitteln: G_i ist im Verhältnis dichter (dünner) als G, wenn der Quotient $w(G_i)/|V_i|^\theta$ größer (kleiner) als 1 ist. Die erwartete Dichte $\bar{\rho}$ für das Referenzclustering $\mathcal{C}^\star$ auf den Ähnlichkeitsgraphen G errechnet sich also wie folgt:

$$\bar{\rho}(\mathcal{C}^\star) = \sum_{i=1}^{l} \frac{|V_i|}{|V|} \cdot \frac{w(G_i)}{|V_i|^\theta} \qquad \text{mit} \quad |V|^\theta = w(G)$$

Je größer $\bar{\rho}$ ist, desto besser spiegelt der Ähnlichkeitsgraph, respektive das verwendete Dokumentmodell, die intrinsischen Ähnlichkeiten der Dokumentkollektion wider.

Literatur

Abouelhoda u. a. 2004. ABOUELHODA, Mohamed Ibrahim ; KURTZ, Stefan ; OHLEBUSCH, Enno: Replacing Suffix Trees with Enhanced Suffix Arrays. In: *Journal of Discrete Algorithms* 2 (2004), Nr. 1, S. 53–86. – ISSN 1570-8667

Andoni und Indyk 2006. ANDONI, Alexandr ; INDYK, Piotr: Efficient algorithms for substring near neighbor problem. In: *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. New York, NY, USA : ACM Press, 2006, S. 1203–1212. – ISBN 0-89871-605-5

Anh und Moffat 2005. ANH, Vo Ngoc ; MOFFAT, Alistair: Inverted Index Compression Using Word-Aligned Binary Codes. In: *Information Retrieval* 8 (2005), Nr. 1, S. 151–166. – ISSN 1386-4564

Baeza-Yates und Ribeiro-Neto 1999. BAEZA-YATES, Ricardo A. ; RIBEIRO-NETO, Berthier A.: *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999. – URL sunsite.dcc.uchile.cl/irbook/. – ISBN 0-201-39829-X

Bahle u. a. 2002. BAHLE, D. ; WILLIAMS, H. E. ; ZOBEL, J.: Efficient Phrase Querying with an Auxiliary Index. In: JÄRVELIN, K. (Hrsg.) ; BEAULIEU, M. (Hrsg.) ; BAEZA-YATES, R. (Hrsg.) ; MYAENG, S. H. (Hrsg.): *Proceedings of the ACM-SIGIR Conference on Research and Development in Information Retrieval*. Tampere, Finland, 2002, S. 215–221

Bawa u. a. 2005. BAWA, Mayank ; CONDIE, Tyson ; GANESAN, Prasanna: LSH forest: self-tuning indexes for similarity search. In: *WWW '05: Proceedings of the 14th international conference on World Wide Web*. New York, NY, USA : ACM Press, 2005, S. 651–660. – ISBN 1-59593-046-9

Buckland und Gay 1994. BUCKLAND, Michael ; GAY, Fredric: The Relationship between Recall and Precision. In: *Journal of the American Society for Information Science* 45 (1994), Nr. 1, S. 12–19. – ISSN 0002-8231

Buhler 2001. BUHLER: Efficient Large-scale Sequence Comparison by Locality-sensitive Hashing. In: *BIOINF: Bioinformatics* 17 (2001)

Burnard 2000. BURNARD, Lou: The British National Corpus Users Reference Guide. (2000). – URL http://www.natcorp.ox.ac.uk/docs/userManual/

Carter und Wegman 1977. CARTER, J. L. ; WEGMAN, Mark N.: Universal classes of hash functions (Extended Abstract). In: *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*. New York, NY, USA : ACM Press, 1977, S. 106–112

Chen 2002. CHEN, Yangjun: Signature Files and Signature Trees. In: *Inf. Process. Lett.* 82 (2002), Nr. 4, S. 213–221. – ISSN 0020-0190

Chen 2005. CHEN, Yangjun: On the General Signature Trees. In: *DEXA*, 2005, S. 207–219

Cormen u. a. 2001. CORMEN, Thomas H. ; LEISERSON, Charles E. ; RIVEST, Ronald L. ; STEIN, Clifford: *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001. – ISBN 0-26253-196-8

Datar u. a. 2004. DATAR, Mayur ; IMMORLICA, Nicole ; INDYK, Piotr ; MIRROKNI, Vahab S.: Locality-sensitive hashing scheme based on p-stable distributions. In: *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*. New York, NY, USA : ACM Press, 2004, S. 253–262. – ISBN 1-58113-885-7

Dutta u. a. 2006. DUTTA, D. ; GUHA, R. ; JURS, P.C. ; CHEN, T.: Scalable Partitioning and Exploration of Chemical Spaces Using Geometric Hashing. In: *Journal of Chemical Information and Modeling* 46 (2006), Nr. 1, S. 321–333

Faloutsos 1992. FALOUTSOS, Christos: Signature files. (1992), S. 44–65. ISBN 0-13-463837-9

Garratt u. a. 2001. GARRATT, Andrea ; JACKSON, Mike ; BURDEN, Peter ; WALLIS, Jon: A survey of alternative designs for a search engine storage structure. In: *Information & Software Technology* 43 (2001), Nr. 11, S. 661–677

Gionis u. a. 1999. GIONIS, Aristides ; INDYK, Piotr ; MOTWANI, Rajeev: Similarity Search in High Dimensions via Hashing. In: *The VLDB Journal*, 1999, S. 518–529

Grossi u. a. 2004. GROSSI, Roberto ; GUPTA, Ankur ; VITTER, Jeffrey Scott: When Indexing Equals Compression: Experiments with Compressing Suffix Arrays and Applications. In: *SODA '04: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 2004, S. 636–645. – ISBN 0-89871-558-X

Gusfield 1997. GUSFIELD, Dan: *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997

Guttman 1984. GUTTMAN, Antonin: R-Trees: A Dynamic Index Structure for Spatial Searching. In: *SIGMOD Conference*, 1984, S. 47–57

Halkidi u. a. 2002. HALKIDI, Maria ; BATISTAKIS, Yannis ; VAZIRGIANNIS, Michalis: Cluster validity methods: part I. In: *SIGMOD Rec.* 31 (2002), Nr. 2, S. 40–45. – ISSN 0163-5808

Haveliwala u. a. 2000. HAVELIWALA, Taher H. ; GIONIS, Aristides ; INDYK, Piotr: Scalable Techniques for Clustering the Web. In: *WebDB (Informal Proceedings)*, 2000, S. 129–134

Heinz und Zobel 2003. HEINZ, Steffen ; ZOBEL, Justin: Efficient single-pass index construction for text databases. In: *Journal of the American Society for Information Science and Technology* 54 (2003), Nr. 8, S. 713–729. – ISSN 1532-2882

Helmer und Moerkotte 2003. HELMER, Sven ; MOERKOTTE, Guido: A performance study of four index structures for set-valued attributes of low cardinality. In: *VLDB J.* 12 (2003), Nr. 3, S. 244–261

Indyk 2000. INDYK, P.: Stable distributions, pseudorandom generators, embeddings and data stream computation. In: *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. Washington, DC, USA : IEEE Computer Society, 2000, S. 189. – ISBN 0-7695-0850-2

Indyk und Motwani 1998. INDYK, Piotr ; MOTWANI, Rajeev: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: *Proc. of 30th STOC*, 1998, S. 604–613

Kim u. a. 2003. KIM, Dong Kyue ; SIM, Jeong Seop ; PARK, Heejin ; PARK, Kunsoo: Linear-Time Construction of Suffix Arrays. In: *CPM*, 2003, S. 186–199

Knuth 1998. KNUTH, Donald E.: *The art of computer programming, volume 3: sorting and searching*. 2nd. Reading : Addison-Wesley, 1998. – ISBN 0–201–89685–0

Ko und Aluru 2003. KO, Pang ; ALURU, Srinivas: Space Efficient Linear Time Construction of Suffix Arrays. In: *CPM*, 2003, S. 200–210

Kärkkäinen und Sanders 2003. KÄRKKÄINEN, J. ; SANDERS, P.: Simple linear work suffix array construction. In: *Proc. 13th International Conference on Automata, Languages and Programming*, Springer, 2003

Kulig 2006. KULIG, Marion: *Intrinsische Plagiatanalyse am Beispiel einer Artikelsammlung*. Bislang unveröffentlichte Diplomarbeit. Erstellt am Institut für Content Management und Web Technologien der Fakultät Mediensysteme an der Bauhaus-Universität Weimar. 2006

Lee u. a. 1995. LEE, Dik L. ; KIM, Young M. ; PATEL, Gaurav: Efficient Signature File Methods for Text Retrieval. In: *IEEE Transactions on Knowledge and Data Engineering* 7 (1995), Nr. 3, S. 423–435. – ISSN 1041-4347

Lee und Park 2004. LEE, Sunglim ; PARK, Kunsoo: Efficient Implementations of Suffix Array Construction Algorithms. In: *Proceedings of the 15th Australasian Workshop on Combinatorial Algorithms*, 2004, S. 64–72

Lewis u. a. 2004. LEWIS, David D. ; YANG, Yiming ; ROSE, Tony G. ; LI, Fan: RCV1: A New Benchmark Collection for Text Categorization Research. In: *J. Mach. Learn. Res.* 5 (2004), S. 361–397. – ISSN 1533-7928

Linial u. a. 1995. LINIAL, Nathan ; LONDON, Eran ; RABINOVICH, Yuri: The Geometry of Graphs and some of its Algorithmic Applications. In: *Combinatorica* 15 (1995), S. 215–245

Manber und Myers 1990. MANBER, Udi ; MYERS, Gene: Suffix Arrays: A new Method for On-Line String Searches. In: *SODA '90: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics, 1990, S. 319–327. – ISBN 0-89871-251-3

Manolopoulos u. a. 2003. MANOLOPOULOS, Yannis ; NANOPOULOS, Alexandros ; PAPADOPOULOS, Apostolos N. ; THEODORIDIS, Yannis: R-Trees Have Grown Everywhere. (2003)

Meyer zu Eissen und Stein 2002. MEYER ZU EISSEN, Sven ; STEIN, Benno: Analysis of Clustering Algorithms for Web-Based Search. In: KARAGIANNIS, Dimitris (Hrsg.) ; REIMER, Ulrich (Hrsg.): *PAKM* Bd. 2569, Springer, 2002, S. 168–178

Meyer zu Eissen u. a. 2005. MEYER ZU EISSEN, Sven ; STEIN, Benno ; POTTHAST, Martin: The Suffix Tree Document Model Revisited. In: TOCHTERMANN, K. (Hrsg.) ; MAURER, H. A. (Hrsg.): *Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 2005)*. Graz, Austria, 2005, S. 596–603. – ISSN 0948-695x

Nicholas Lester u. a. 2005. NICHOLAS LESTER ; MOFFAT, Alistair ; ZOBEL, Justin: Fast On-Line Index Construction by Geometric Partitioning. In: *CIKM '05: Proceedings of the 14th ACM International Conference on Information and Knowledge Management*. New York, NY, USA : ACM Press, 2005, S. 776–783. – ISBN 1-59593-140-6

Nolan 2005. NOLAN, John P.: *Stable Distributions—Models for Heavy Tailed Data*. http://academic2.american.edu/~jpnolan/stable/stable.html. 2005

Porter 1980. PORTER, M. F.: An Algorithm for Suffix Stripping. In: *Program* 14 (1980), Nr. 3, S. 130–137

Postel 2006. POSTEL, Jon: *RFC Collection*. http://www.rfc-editor.org/. 2006

Salton und Lesk 1968. SALTON, G. ; LESK, M. E.: Computer Evaluation of Indexing and Text Processing. In: *Journal of the ACM* 15 (1968), Nr. 1, S. 8–36. – ISSN 0004-5411

Shakhnarovich u. a. 2003. SHAKHNAROVICH, Gregory ; VIOLA, Paul ; DARRELL, Trevor: Fast Pose Estimation with Parameter-Sensitive Hashing. In: *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*. Washington, DC, USA : IEEE Computer Society, 2003, S. 750. – ISBN 0-7695-1950-4

Stein 2005a. STEIN, Benno: Fuzzy-Fingerprints for Text-Based Information Retrieval. In: *Proc. of the I-KNOW*, 24–27 2005, S. 194–205

Stein 2005b. STEIN, Benno: *Lecture on Advanced Web Technology*. http://www.uni-weimar.de/cms/Lecture_ Notes.550.0.html. 2005

Stein u. a. 2003. STEIN, Benno ; MEYER ZU EISSEN, Sven ; WISSBROCK, Frank: On Cluster Validity and the Information Need of Users. In: HANZA, M. H. (Hrsg.): *Proceedings of the 3rd IASTED International Conference on Artificial Intelligence and Applications (AIA 03), Benalmádena, Spain*. Anaheim, Calgary, Zurich : ACTA Press, September 2003, S. 216–221. – ISBN 0-88986-390-3

Stein und Niggemann 1999. STEIN, Benno ; NIGGEMANN, Oliver: On the Nature of Structure and Its Identification. In: *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science* Bd. 1665, Springer, 1999, S. 122–134. – ISBN 3-540-66731-8

Stein und Potthast 2006. STEIN, Benno ; POTTHAST, Martin: *Putting Successor Variety Stemming to Work*. 2006

Tan u. a. 2005. TAN, Pan-Ning ; STEINBACH, Michael ; KUMAR, Vipin: *Introduction to Data Mining*. Addison Wesley, 2005

Tian u. a. 2005. TIAN, Yuanyuan ; TATA, Sandeep ; HANKINS, A. ; PATEL, M.: Practical Methods for Constructing Suffix Trees. In: *The VLDB Journal* 14 (2005), Nr. 3, S. 281–299. – ISSN 1066-8888

Trotman 2003. TROTMAN, Andrew: Compressing Inverted Files. In: *Information Retrieval* 6 (2003), Nr. 1, S. 5–19

Ukkonen 1995. UKKONEN, Esko: On-Line Construction of Suffix Trees. In: *Algorithmica* 14 (1995), Nr. 3, S. 249–260

Weber u. a. 1998. WEBER, Roger ; SCHEK, Hans-Jörg ; BLOTT, Stephen: A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In: *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, 24–27 1998, S. 194–205

Williams u. a. 1999. WILLIAMS, Hugh E. ; ZOBEL, Justin ; ANDERSON, Phil: What's Next? Index Structures for Efficient Phrase Querying. In: *Australasian Database Conference*, 1999, S. 141–152

Witten u. a. 1999. WITTEN, I. H. ; MOFFAT, A. ; BELL, T. C.: *Managing Gigabytes (2nd Ed.): Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers Inc., 1999. – ISBN 1-55860-570-3

Zobel u. a. 1998. ZOBEL, Justin ; MOFFAT, Alistair ; RAMAMOHANARAO, Kotagiri: Inverted files versus signature files for text indexing. In: *ACM Trans. Database Syst.* 23 (1998), Nr. 4, S. 453–490. – ISSN 0362-5915