Grundlagen des Multimediaretrievals

Vorlesung im Wintersemester 2005/2006

Prof. Dr. Michael Clausen
Priv Doz. Dr. Frank Kurth
Dr. Meinard Müller

Institut für Informatik III
Römerstraße 164
Rheinische Friedrich-Wilhelms-Universität Bonn

Version vom 13. März 2006

Vorwort zur Vorlesung im Wintersemester 2005/2006

Die vorliegende Version des Skriptums umfaßt den Stoff der Vorlesung *Grundlagen des Multimediaretrievals* aus dem Wintersemester 2005/2006, der sich im wesentlichen an die Inhalte der beiden Vorgängervorlesungen aus den Wintersemestern 2002/2003 und 2003/2004 anlehnt.

Für zahlreiche Verbesserungsvorschläge und Korrekturen möchten wir uns insbesondere bei den Hörern der Vorlesungen herzlich bedanken.

Bonn, 13. März 2006

Michael Clausen
Frank Kurth
Meinard Müller

Vorwort

Dieses Skriptum ist aus der Vorlesung *Grundlagen des Multimediaretrievals I* hervorgegangen, die wir jeweils in den Wintersemestern 2002/2003 und 2003/2004 am Institut für Informatik der Universität Bonn gehalten haben. Unser Ziel bei dieser Vorlesung war es, sowohl eine angemessene Einführung in wichtige Aspekte des klassischen *Textretrievals* zu geben (Kapitel 1), als auch das Retrieval auf allgemeinen (nicht-Text- bzw. multimedialen) Dokumententypen einführend zu betrachten. Der hierzu eingeschlagene Weg führt uns nach dem Textretrieval zunächst auf das in den letzten Jahren so praxisrelevant gewordene *Webretrieval* (Kapitel 2). In diesem Kapitel stellen insbesondere die in der Suchmaschine `Google` eingesetzten Ranking-Verfahren einen wichtigen Beitrag zur Ordnung der allgegenwärtigen WWW-Datenflut dar. Die Betrachtung der "multimedialen" Dokumententypen konzentriert sich dann ab Kapitel 3 zunächst — exemplarisch — auf das (symbolische) *Musikretrieval*. Hier haben wir eine von vielen derzeit bekannten Retrievalstrategien herausgegriffen und im Detail beleuchtet. Die Tragweite dieser Retrieval-Strategie läßt sich mit Methoden der elementaren Gruppentheorie beschreiben, was Gegenstand des abschließenden Kapitels 4 über ein *Allgemeines Retrievalkonzept* ist.

Die vorliegende Version des Skriptums umfaßt den vollständigen Stoff der Vorlesung aus dem Wintersemester 2003/2004. Im Vergleich zur ersten Veranstaltung von 2002/2003 ist hier insbesondere eine überarbeitete Version der Kapitel 3 und 4 enthalten. Das Skriptum ist im Verlauf der beiden Veranstaltungen gewachsen und hat, nach unserer Meinung, durch zahlreiche Verbesserungsvorschläge unserer Studierenden immer weiter an Qualität gewonnen. Für dieses Feedback möchten wir uns an dieser Stelle herzlich bedanken.

Bonn, 22. März 2005

Michael Clausen
Frank Kurth

Inhaltsverzeichnis

Kapitel 1

Grundtechniken am Beispiel des Textretrievals

1.1 Datenmodellierung

Zunächst stellen wir die grundsätzliche Frage, was wir unter einem Textdokument verstehen wollen

Klassische Sicht:

Gegeben ist ein (endliches) *Alphabet* Σ und eine *Sprache*

$$L \subset \Sigma^+ := \bigcup_{m \geq 1} \Sigma^m.$$

Σ^+ ist die Menge aller endlichen Wörter über Σ. Ein *Textdokument* kann nun modelliert werden

1. als Wortfolge $(t_1, \ldots, t_n) \in L^n$ oder
2. als Zeichenkette $s \in (\Sigma \dot{\cup} \underbrace{\{\sqcup, ., ;, \ldots\}}_{=:\Sigma_0})^+$

   Sonderzeichen

$$s = (t_1 s_1 t_2 s_2 \ldots t_n s_n), \text{ wobei } t_i \in L, s_i \in \Sigma_0^* := \bigcup_{m \geq 0} \Sigma_0^m.$$

Modernere Auffassung:

Textdokumente werden als Konstellation von

- Wörtern
- Sonderzeichen

- Schriftgröße
- Schrifttype und
- Layout

aufgefaßt. Hier gehen wir zunächst nur auf die klassische Sicht ein. Die modernere Sicht wird dem 2. Kapitel zum Thema *Web-Retrieval* zugrunde liegen.
Manche Dokumente müssen *vorverarbeitet* werden, um zu eigentlichen Textdokumenten zu werden:

$$\left.\begin{array}{l}\text{HTML}\\ \text{PDF}\\ \text{Postscript}\end{array}\right\} \longrightarrow \text{Text-Dokument}$$

Auf diese Art von Vorverarbeitung gehen wir hier jedoch nicht näher ein.

1.2 Suchaufgaben, Trefferbegriffe, Retrievalmodelle

1.2.1 Stringbasierte Suche

Einfache klassische Suchaufgaben sind stringbasiert:

- **Stringmatching** (exakt)

  Gegeben: $(D_1, \ldots, D_N)$, eine Sammlung von N Text-Dokumenten. Ein Dokument D_i wird modelliert als Zeichenkette $D_i \in (\Sigma \cup \Sigma_0)^+$.

  Anfrage: $Q \in (\Sigma \cup \Sigma_0)^+$.

  Gesucht sind alle Paare (i, j) bestehend aus einer Dokumentennummer i und einer Positionsangabe j, so dass

  $$Q = (D_i[j], \ldots, D_i[j + |Q| - 1]) =: D_i[j : j + |Q| - 1].$$

- **Stringmatching** (k-Mismatches)

  Gegeben: $(D_1, \ldots D_N)$, eine Sammlung von N Text-Dokumenten wie oben.

  Anfrage: $Q \in (\Sigma \cup \Sigma_0)^+$ wie oben, sowie ein *Fehlertoleranzparameter* $k \in \mathbb{N}_0$.

  Gesucht sind wieder alle Paare (i, j) mit der Eigenschaft, dass

  $$Q \text{ mit } (D_i[j], \ldots, D_i[j + |Q| - 1])$$

  in mindestens $|Q| - k$ Positionen übereinstimmt.

- **Stringmatching** (Editierdistanz)

  Gegeben: Textdokumente $(D_1, \ldots, D_N)$ wie oben.

  Anfrage: $Q \in (\Sigma \cup \Sigma_0)^+$ wie oben, sowie eine *Editierdistanzschranke* $d \in \mathbb{N}_0$.

  Gesucht sind wieder alle Tripel (i, j, ℓ) mit

  $$\mathrm{Edit}(Q, D_i[j : j + \ell]) \leq d.$$

  Hierbei bezeichnet $\mathrm{Edit}(x, y)$ für zwei Strings $x, y \in (\Sigma \cup \Sigma_0)^*$ die minimale Anzahl benötigter Editieroperationen, um einen der Strings in den anderen zu überführen. Dabei sind die Editieroperationen

  – Einfügen (insert),
  – Löschen (delete) und
  – Ändern (replace)

  erlaubt, wobei meist die Kosten für jede dieser Operationen mit 1 festgesetzt werden.

  1.1 Beispiel Seien $x = aabac$ und $y = aabd$ gegeben, dann überführt die Editiersequenz

  $$aabac \overset{\text{Lösche } c}{\vdash} aaba \overset{\text{Ändere letztes } a \to d}{\vdash} aabd$$

  x in zwei Schritten in y. In diesem Fall geht dies auch nicht schneller, also gilt $\mathrm{Edit}(x, y) = 2$. ◇

  Näheres zur Editierdistanz, siehe Übungen (sowie Übungsblätter 1 und 2).

1.2.2 Klassische Retrievalmodelle

Wir besprechen nun klassische Retrievalmodelle. Die Grundidee klassischer IR-Systeme ist, dass die Semantik eines Dokuments sowie das Informationsbedürfnis von Benutzern hinreichend gut durch eine Menge von *Termen* spezifiziert werden kann. Die Menge zulässiger Terme ist eine stark verkleinerte Teilmenge aller möglichen Wörter aus L. Beispiel für Terme und Termbildung:

- Keywords
- Gruppen verwandter Wörter werden zu einem Term zusammengefasst (Bildung sog. *Kontextklassen*). Beispielsweise könnten die Wörter Haus, Villa, Gebäude und Halle zu einem Term `Haus` zusammengefaßt werden.

- Stammformreduktion, z.B. geht → gehen, ging → gehen.

Durch die Einschränkung auf solch eine Termmenge erreicht man einen gewissen Grad an Toleranz, da die Anfrage mit den Dokumenten nur noch bezüglich der Terme übereinstimmen muss und nicht bezüglich der in den Dokumenten vorkommenden Wörter. Eine Umwandlung von Wörtern in Terme wird mit Hilfe eines sogenannten *Termextraktors* vollzogen. Wir werden an späterer Stelle noch auf konkrete Termextraktoren eingehen.

1.2.3 Boolesches Retrievalmodell

Gegeben: Dokumente $(D_1, \ldots, D_N) =: \mathcal{D}$, wobei ein Dokument D_i eine Folge von Wörtern $D_i = (t_{i1}, \ldots, t_{in_i}) \in L^{n_i}$ ist. Weiterhin sei ein Termextraktor

$$\tau : L \longrightarrow T \subseteq L \cup \{\uparrow\}$$

gegeben. Da die Termextraktion nicht für alle Wörter aus $w \in L$ definiert sein muss, verwenden wir die Konvention, dass im undefinierten Fall $\tau(w) :=\uparrow$ gesetzt wird.

Vorverarbeitung: $\tau(D_i) := \{\tau(t_{ij}) \mid 1 \leq j \leq n_i\}\backslash\{\uparrow\}$ ordnet dem Dokument D_i (Folge) eine *Menge* von Termen zu. Hierbei wird also zusätzlich zur Vergröberung von Wörtern auf Terme noch weiter vergröbert, indem die Vielfachheiten der Wörter im Dokument und deren genaue Positionen weggelassen werden. Die Menge aller „relevanten“ Terme, die in mindestens einem Dokument der gesamten Dokumentensammlung vorkommen, erhalten wir durch

$$\begin{aligned} T(\mathcal{D}) &:= \bigcup_{i=1}^{N} \tau(D_i) \\ &=: \{t_1, \ldots, t_M\}. \end{aligned}$$

Wir ordnen jedem Dokument D_i einen booleschen Vektor

$$b(D_i) := (b_{i1}, \ldots, b_{iM}) \in \{0,1\}^M$$

zu, mit

$$b_{ij} := \begin{cases} 1 & \text{falls } t_j \in \tau(D_i) \\ 0 & \text{sonst.} \end{cases}$$

Wir spezifizieren nun die Menge $Q(T)$ aller gültigen booleschen Anfragen, die auf der Termmenge $T \subseteq L$ basieren. $Q(T)$ sei die kleinste Menge mit

- $T \subseteq Q(T)$ und
- $p, q \in Q(T) \Rightarrow \{(p \wedge q), (p \vee q), (\neg p)\} \subseteq Q(T)$.

Ein Element aus $Q(T)$ heißt *boolesche Anfrage* (über T). Man darf dabei also alle Terme aus T anfragen und diese auch (wie gewohnt) logisch verknüpfen.

1.2 Beispiel Bei einer Anfrage

$$((\text{Haus} \ \vee \ \text{Auto}) \wedge (\neg \ \text{Schiff}))$$

sind beispielsweise alle Dokumente gesucht, die (nach Termextraktion) die Terme „Haus“ oder „Auto“, aber nicht den Term „Schiff“ enthalten. ◇

Die **Treffermenge** zu einer Anfrage $q \in Q(T)$ kann nun bezüglich des Termextraktors τ rekursiv definiert werden:

$$B_{\mathcal{D}}(q) := \begin{cases} \{i \mid q \in \tau(D_i)\} & \text{falls } q \in T \\ B_{\mathcal{D}}(p) \cup B_{\mathcal{D}}(r) & \text{falls } q = (p \vee r), \\ B_{\mathcal{D}}(p) \cap B_{\mathcal{D}}(r) & \text{falls } q = (p \wedge r), \\ [1:N] \backslash B_{\mathcal{D}}(p) & \text{falls } q = (\neg p). \end{cases}$$

Die Semantik des booleschen Retrievals wird also durch eine Abbildung $B_{\mathcal{D}} : Q(T) \to 2^{[1:N]}$ beschrieben, die einer Anfrage q die Menge $B_{\mathcal{D}}(q)$ alle Dokumentennummern zuordnet, die die Anfrage erfüllen.

Bei folgender alternativer Sichtweise fragt man, ob ein Dokument D_i zu einer Anfrage $q \in Q(T)$ einen Treffer liefert ($\equiv 1$) oder nicht ($\equiv 0$). Die zugehörige Abbildung

$$B'_{\mathcal{D}} : Q(T) \times [1:N] \to \{0,1\}$$

wobei

$$B'_{\mathcal{D}}(q,i) := \begin{cases} 1 & \text{falls } q \in T \cap \tau(D_i), \\ 0 & \text{falls } q \in T \backslash \tau(D_i), \\ \max\{B'_{\mathcal{D}}(p,i), B'_{\mathcal{D}}(r,i)\} & \text{falls } q = (p \vee r), \\ \min\{B'_{\mathcal{D}}(p,i), B'_{\mathcal{D}}(r,i)\} & \text{falls } q = (p \wedge r), \\ 1 - B'_{\mathcal{D}}(p,i) & \text{falls } q = (\neg p). \end{cases}$$

wird oft auch als *rsv (retrieval status value)* bezeichnet.

Vorteile des booleschen Retrievalmodells:

- Man kann genau spezifizieren, wonach man sucht.
- Eine sehr effiziente Implementierung der Retrievalfunktionen $B_{\mathcal{D}}$ bzw. $B'_{\mathcal{D}}$ ist möglich ($\to$ invertierte Listen).

Nachteile des booleschen Retrievalmodells:

- Innerhalb dieses Modells gibt es nur eine binäre Entscheidung in *relevante* und *nicht relevante* Dokumente bzgl. einer Anfrage.
- Es sind keine partiellen Treffer (nur Teile des Ausdrucks matchen) möglich.
- Boolesches Retrieval liefert kein Ranking der (möglicherweise großen) Trefferliste. Dies führt oft zu schlechter Retrievalqualität.
- Die Anfragesprache ist oft zu komplex für Endbenutzer.

1.2.4 Fuzzy-Retrieval

Eine Möglichkeit, weg von rein binären Relevanzentscheidungen zu kommen, bietet das *Fuzzy-Retrieval*. Hierzu wiederholen wir zunächst einige grundlegende Fakten über sogenannte Fuzzymengen. Vorab sei davor gewarnt, dass der Begriff *Menge* in diesem Zusammenhang zwar allgemein gebräuchlich ist, aber trotzdem für Verwirrungen sorgen kann, da die hier betrachteten Fuzzymengen in Wirklichkeit *Abbildungen* sind.

1.3 Definition Sei U eine Menge ($U \equiv$ Universum). Eine *Fuzzyteilmenge* von U ist eine Abbildung

$$\mu : U \to [0,1] \subset \mathbb{R}.$$

Für $u \in U$ beschreibt $\mu(u)$ den Grad der Zugehörigkeit von u zu μ. ◇

1.4 Bemerkung Betrachtet man den Grenzfall $\mu : U \to \{0,1\}$, erhält man die klassische Mengensichtweise; μ ist dann die Indikatorfunktion der Teilmenge $\{u \in U \mid \mu(u) = 1\}$. ◇

Auf Fuzzyteilmengen werden nun die „Mengenoperationen“ Komplementbildung, Vereinigung und Durchschnitt definiert. Aus jeder dieser Operationen resultiert wieder eine Fuzzyteilmenge (Funktion).

1.5 Definition Seien μ und ν Fuzzyteilmengen von U. Dann definieren wir folgende Fuzzyteilmengen:

$\overline{\mu} : u \mapsto 1 - \mu(u)$ (Komplement),

$(\mu \cup \nu) : u \mapsto \max(\mu(u), \nu(u))$ (Vereinigung),

$(\mu \cap \nu) : u \mapsto \min(\mu(u), \nu(u))$ (Schnitt).

◇

Hierzu gibt es folgende algebraische Varianten:

$(\mu \cup' \nu) : u \mapsto \mu(u) + \nu(u) - \mu(u) \cdot \nu(u)$ (Vereinigung),

$(\mu \cap' \nu) : u \mapsto \mu(u) \cdot \nu(u)$ (Schnitt).

Beachte: $(\mu \cup \nu) \leq (\mu \cup' \nu)$ und $(\mu \cap \nu) \geq (\mu \cap' \nu)$.
Die nun vorgestellte Idee zum Fuzzy-Retrieval wurde von Ogawa, Morita und Kobayashi vorgeschlagen. Zunächst wird anhand einer *Term-Term Korrelationsmatrix* eine Art Thesaurus gebildet. Seien $t_1, \ldots, t_M$ die relevanten Terme in der Dokumentensammlung $\mathcal{D} = (D_1, \ldots, D_N)$.

Die Term-Term Korrelationsmatrix $C = C_{\mathcal{D}} = (c_{i\ell}) \in \mathbb{Q}^{M \times M}$ ist dann definiert durch

$$c_{i\ell} := \frac{\#\{j | \{t_i, t_\ell\} \subseteq \tau(D_j)\}}{\#\{j | \{t_i, t_\ell\} \cap \tau(D_j) \neq \emptyset\}} \text{ „}=\text{“ } \frac{\#_{\mathcal{D}}(i \wedge \ell)}{\#_{\mathcal{D}}(i \vee \ell)},$$

wobei zu beachten ist, dass $\#_{\mathcal{D}} i \vee \ell \neq 0$ gilt, $c_{i\ell}$ also wohldefiniert ist.

Mit Hilfe dieser Korrelationsmatrix wird nun für jeden Term t_i eine Fuzzyteilmenge μ_i über dem Universum $[1:N]$ der Dokumentennummern definiert:

$$\mu_{ij} := \mu_i(j) := 1 - \prod_{t_\ell \in \tau(D_j)} (1 - c_{i\ell}), \text{ für } 1 \leq j \leq N.$$

Dabei beschreibt $\mu_{i,j}$ den Grad der Zugehörigkeit von Dokument D_j zum Term t_i: Gibt es im Dokument D_j einen Term t_ℓ, der mit t_i stark korreliert (d.h. $c_{i\ell} \sim 1$), so ist das Produkt fast Null, also $\mu_{ij} \sim 1$. Sind alle Terme t_ℓ in D_j wenig korreliert mit t_i (d.h. $\forall t_\ell \in \tau(D_j) : c_{i\ell} \sim 0$), so ist das Produkt groß, also μ_{ij} klein.

Anfragen beim Fuzzy-Retrieval haben die gleiche Form wie beim booleschen Retrieval. Die **Auswertung** geschieht analog zum booleschen Retrieval unter Verwendung einer Retrievalfunktion $F_\mathcal{D}$ ähnlich zu $B'_\mathcal{D}$. Der Unterschied besteht darin, dass beim Fuzzy-Retrieval Werte in $[0,1]$ angenommen werden können:

$$F_\mathcal{D} : Q(T) \times [1:N] \to [0,1],$$

wobei

$$F_\mathcal{D}(q,j) := \begin{cases} \mu_{ij} & \text{falls } q = t_i, \\ F_\mathcal{D}(p,j) \cdot F_\mathcal{D}(r,j) & \text{falls } q = (p \wedge r) \\ F_\mathcal{D}(p,j) + F_\mathcal{D}(r,j) - F_\mathcal{D}(p,j) \cdot F_\mathcal{D}(r,j) & \text{falls } q = (p \vee r) \\ 1 - F_\mathcal{D}(p,j) & \text{falls } q = (\neg p). \end{cases}$$

Beachte, dass hier die algebraischen Varianten für Vereinigung und Schnittbildung von Fuzzyteilmengen zugrunde liegen.
Abschließend ist zu bemerken, dass es noch einige weitere Varianten zum Thema Fuzzy-Retrieval gibt.

Vorteile des Fuzzy-Retrievals:

- Ordnung der gefundenen Dokumente möglich (Ordnung der $(F_\mathcal{D}(q,j))_j$ der Größe nach, wobei höhere $F_\mathcal{D}$-Werte eine höhere Relevanz des Trefferkandidaten angeben),
- Term-Term Korrelationen werden berücksichtigt.

Nachteile des Fuzzy-Retrievals:

- Die Ergebnisse sind nicht wesentlich besser als beim booleschen Retrieval und schlechter als beim im folgenden beschriebenen Vektorraum-Retrieval,
- wie beim booleschen Retrieval ist die Anfragesprache oft zu komplex für Endbenutzer.

1.2.5 Vektorraumretrieval

Idee: Dokumente und Anfragen werden durch Vektoren im $\mathbb{R}^M$ ($M := |T|$ ist hier die Anzahl der benutzten Terme) beschrieben. Mit Hilfe einer geeigneten Metrik werden dann Abstände zwischen den Dokumenten und einer Anfrage berechnet. Dies liefert ein Ranking der Dokumente bezüglich der Anfrage.

Das **Ziel** ist eine Zuordnung

$$D_j \mapsto \begin{pmatrix} w_{1j} \\ \vdots \\ w_{Mj} \end{pmatrix} \in \mathbb{R}^M$$

eines Dokuments D_j zu einem Vektor in $\mathbb{R}^M$, wobei das Gewicht w_{ij} misst, wie stark sich D_j innerhalb der Kollektion $(D_1, \ldots, D_N)$ aufgrund des Auftretens des i-ten Terms t_i abgrenzt.
In die Berechnung von w_{ij} gehen ein

- die *normalisierte Termhäufigkeit*

  $$f_{ij} := \frac{\#\text{ Vorkommen von } t_i \text{ in } D_j}{\max_\ell \#\text{ Vorkommen von } t_\ell \text{ in } D_j} \geq 0$$

  wobei für $D_j = (t_{j1}, \ldots, t_{jn_j})$

  $$\text{“\# Vorkommen von } t_i \text{ in } D_j\text{“} := \#\{k \in [1 : n_j] \,|\, t_i = \tau(t_{jk})\}.$$

- die *Dokumentenhäufigkeit*

  $$\mathrm{df}_i := \#\{j \mid t_i \in \tau(D_j)\}$$

  (engl. *document frequency*) und

- die *inverse Dokumentenhäufigkeit*

  $$\mathrm{idf}_i := \log \frac{N}{\mathrm{df}_i} \geq 0$$

  (engl. *inverse document frequency*).

1.6 Bemerkung Die inverse Dokumentenhäufigkeit soll für jeden Term t_i ein Gewicht berechnen, welches grob ausdrückt, wie „deskribierend“ t_i in der Kollektion $(D_1, \ldots, D_N)$ ist. Ein Term t_i, der z.B. in *jedem* Dokument vorkommt, eignet sich nicht zum Trennen von Dokumenten. In diesem Fall ist $|\{j \mid t_i \in \tau(D_j)\}| = \mathrm{df}_i = N$, also $\mathrm{idf}_i = \log 1 = 0$. Besser zum Trennen von Dokumenten sind Terme, die nur sehr selten vorkommen. Abbildung 1.1 skizziert idf gegen df für $N = 100$. ◇

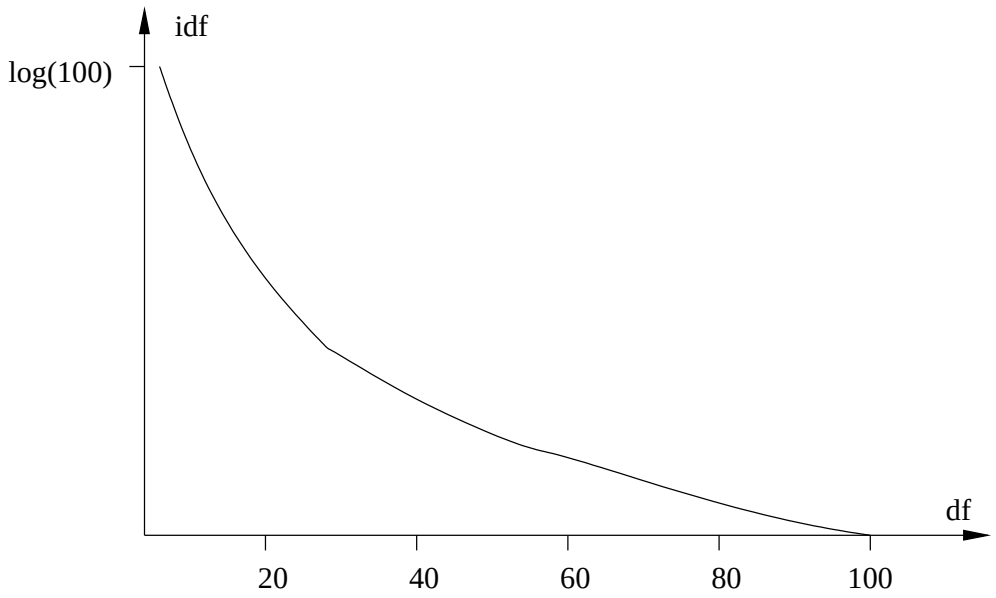


Abbildung 1.1: idf gegen df für $N = 100$.

Der Gewichtsvektor $(w_{ij}) \in \mathbb{R}^M$ zu D_j wird nun definiert durch

$$w_{ij} := f_{ij} \cdot \text{ idf}_i \geq 0.$$

Eine Anfrage q wird auch als Dokument behandelt und erhält analog einen Gewichtsvektor $(w_{1q}, \ldots, w_{Mq})^\top \in \mathbb{R}^M$. Zur Gewichtung der Anfrage q schlagen Salton und Buckley (siehe *Information Processing and Management*, Band 24 (5), Seiten 513–523, 1988) folgende Modifikation vor:

$$w_{iq} := \frac{1}{2}\,(1 + f_{iq}) \cdot \text{ idf}_i.$$

Beachte, dass somit alle Gewichte ≥ 0 sind.

Auswertung beim Vektorraum-Retrieval:

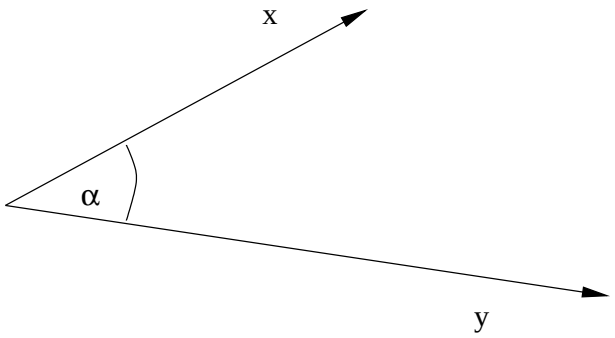
- Die *Retrieval-Funktion* ordnet jedem Paar (q, D_j) aus Anfrage und Dokument eine Relevanzzahl ≥ 0 zu, den sogenannten Retrieval Status Value $\text{rsv}(q, D_j)$ (vgl. auch die vorherigen Anmerkungen beim booleschen Retrieval).

- Bei einer Anfrage q werden die Dokumente D_j nach absteigender Relevanzzahl geordnet und die ersten k Dokumente zurückgegeben.

Gebräuchliche **Retrievalfunktionen:**

- $\text{rsv}(q, D_j) := \langle (w_{iq}) | (w_{ij}) \rangle = \sum_{i=1}^{M} w_{iq} w_{ij}$ (inneres Produkt).

- Memo: $\langle x|y\rangle = \|x\| \cdot \|y\| \cdot \cos\alpha$



(Interpretation des normalisierten inneren Produktes als Winkelmaß.) Das normalisierte innere Produkt

$$\mathrm{RSV}(q, D_j) = \frac{\langle (w_{iq})|(w_{ij})\rangle}{\|(w_{iq})\| \cdot \|(w_{ij})\|} = \text{„cos } \alpha\text{“}$$

heißt *Kosinus-Maß*. Hier ist $\mathrm{RSV}(q, D_j) \in [0, 1]$, denn alle Gewichte sind ≥ 0.

1.7 Bemerkung

- Es gibt viele weitere Varianten für Retrievalfunktionen und Gewichtungen.
- Vektorraumretrieval basiert auf der Annahme, dass die Terme unabhängig sind. Dies ist offensichtlich i.a. nicht zutreffend, z.B. *Auto*, *BMW*, *Mercedes*, *VW*, ... Die Term-Term Korrelationsmatrix beim Fuzzy-Retrieval hat hier versucht, diese Abhängigkeiten zu berücksichtigen.
- Eine weiteres auf dem Vektorraummodell basierendes Retrievalmodell ist das *Latent Semantic Indexing (LSI)*: Mit Methoden der linearen Algebra wird versucht, die Abhängigkeiten zwischen Termen explizit zu machen und die Dimension des Termraums durch Zusammenführen von Termen zu Konzepten stark zu reduzieren (siehe Übungen).

◇

Vorteile des Vektorraumretrievals:

- Das Modell ist einfach und die rsv's sind i.a. effizient auswertbar,
- ein *partial match* ist möglich, d.h. es werden auch Dokumente gefunden, die nur einige der Anfrageterme enthalten,
- das Modell liefert eine gute Retrievalqualität,
- Relevanz-Feedback einfach integrierbar (siehe später).

Nachteile des Vektorraumretrievals:

- Viele Heuristiken und Vereinfachungen werden verwendet (z.B. Termhäufigkeiten und Gewichtungen),
- Layoutinformationen werden nicht berücksichtigt (wie z.B. Fettdruck, Überschriften,...).

1.2.6 Probabilistisches Retrieval

Gegeben seien

- eine Dokumentenkollektion $\mathcal{D} = (D_1, \ldots, D_N)$, wo $D_j \in L^+$,
- eine Anfrage $Q \in L^+$ und
- ein Termextraktor $\tau : L \to T \subset L$, also $\tau(D_j) \subset T$ und $\tau(Q) \subset T$.

Ziel: Teile die Menge $[1 : N]$ der Dokumenten-IDs disjunkt auf in einen Q-relevanten Teil R und einen Q-irrelevanten Teil $\overline{R}$:

$$[1 : N] = R \,\dot\cup\, \overline{R}.$$

Annahme: $R = R(Q, \mathcal{D}, \tau)$, d.h. R hängt ab von der Anfrage Q, der Dokumentenkollektion $\mathcal{D}$ und dem Termextraktor τ. Jedoch ist R unabhängig vom Benutzer (natürlich i.a. unrealistisch).

Frage: Wie kommt man an R?

Idee: Ordne jedem D_j bzgl. Q und τ eine Wahrscheinlichkeit zu, dass D_j bzgl. Q relevant ist. Dies geschieht *iterativ*, indem man mit einer Wahrscheinlichkeitsverteilung (W-Verteilung) auf $[1 : N]$ beginnt und dann durch *Iteration* zwischen System und Benutzer („Relevanz-Feedback“) die W-Verteilung auf $[1 : N]$ sukzessive verbessert.

Herleitung und Motivation der Ausgangsverteilung: Dokumenten und Anfrage wird durch Termextraktion jeweils eine Menge von Termen zugeordnet:

$$\tau(Q), \tau(D_1), \ldots, \tau(D_N).$$

Die Idee ist nun, die Relevanzentscheidung wesentlich von der Verteilung der Anfrageterme $t_i \in \tau(Q)$ auf die Dokumententermmengen $\tau(D_j)$ abhängig zu machen. Sei

$$N_i := \{j \in [1 : N] \mid t_i \in \tau(D_j)\}$$

die Menge derjenigen Dokumentennummern, in denen Term t_i (nach Termextraktion) vorkommt. Ein Dokument, das $t_i \in \tau(Q)$ enthält, steht bezüglich der Anfrage Q mit allen Dokumenten D_j mit $j \in N_i$ in Konkurrenz. Analog steht ein Dokument D_ℓ, das $t_i \in \tau(Q)$ *nicht* enthält mit allen D_j mit $j \in [1 : N] \backslash N_i = \overline{N_i}$,

in Konkurrenz. Indem man dies für alle Terme $t_i \in \tau(Q)$ simultan berücksichtigt, stößt man auf die Menge

$$N_Q(D_j) := \left(\bigcap_{t_i \in \tau(Q) \cap \tau(D_j)} N_i \right) \cap \left(\bigcap_{t_i \in \tau(Q) \setminus \tau(D_j)} \overline{N_i} \right),$$

wobei $\bigcap_{t_i \in \emptyset} N_i := [1 : N]$. Diese Teilmenge von $[1 : N]$ charakterisiert das j-te Dokument aus Sicht der Termmengen. Insbesondere gilt

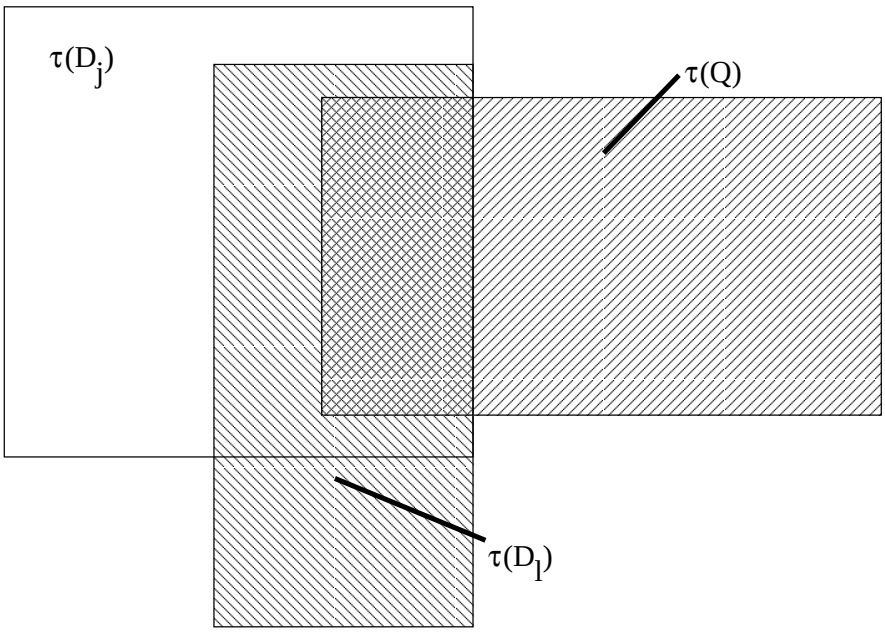
$$j \in N_Q(D_j).$$

Es ist zu beachten, dass Dokumente D_j und D_ℓ mit

$$\tau(Q) \cap \tau(D_j) = \tau(Q) \cap \tau(D_\ell) \tag{1.1}$$

aus Anfragesicht nicht unterscheidbar sind. Genauer sind die $N_Q(D_j)$ die Äquivalenzklassen der Äquivalenzrelation

$$j \sim_Q \ell :\Leftrightarrow \tau(Q) \cap \tau(D_j) = \tau(Q) \cap \tau(D_\ell).$$



Analyse unter der Annahme, dass die W-Verteilung P und die Menge R der Q-relevanten Dokumenten-IDs bekannt sind: Sei

$$P(R|D_j) := P(R|N_Q(D_j))$$

die Wahrscheinlichkeit, dass D_j Q-relevant ist und

$$P(\overline{R}|D_j) := P(\overline{R}|N_Q(D_j))$$

die Wahrscheinlichkeit, dass D_j Q-irrelevant ist.
Anmerkung: Die Schreibweisen $P(R|D_j)$ und $P(\overline{R}|D_j)$ sind sehr schlampig, da D_j ein *Dokument* und keine *Menge* bezeichnet. Schlimmer noch: Nicht einmal die „intuitiv" zu D_j korrespondierende Menge $\{j\}$ bestehend aus der Dokumentennummer liefert hier den gewünschten Ausdruck, da i.a. von $\{j\} \neq N_Q(D_j)$ ausgegangen werden muss. Leider ist diese Schreibweise aber in der Literatur sehr verbreitet, weswegen wir sie an dieser Stelle verwenden.
Wir definieren

$$\mathrm{sim}(D_j, Q) := \frac{P(R|D_j)}{P(\overline{R}|D_j)}$$

als Maß für die *Ähnlichkeit* des Dokuments D_j bezüglich der Anfrage Q. Dokumente werden dann schließlich nach absteigender Ähnlichkeit zur Anfrage geordnet. Zur Berechnung der Ähnlichkeit wiederholen wir zunächst einige Fakten aus der Stochastik.
Die *bedingte Wahrscheinlichkeit* (s.o.) für zwei Mengen A und B aus der Ereignisalgebra (hier $2^{[1:N]}$) und eine W-Verteilung P ist, sofern $P(B) > 0$, definiert als

$$P(A|B) := \frac{P(A \cap B)}{P(B)}.$$

Aus $P(A|B)$ und $P(B|A)$ ergibt sich trivialerweise der
1.8 Satz (Bayes)

$$P(A|B) \cdot P(B) = P(A \cap B) = P(B|A) \cdot P(A).$$

◇

Dieser Satz wird sehr oft verwendet. Intuitiv erlaubt er, „Ursache" und „Wirkung" zu vertauschen, d.h. kennt man $P(A|B)$ (sowie $P(A)$ und $P(B)$), kann man auf $P(B|A)$ schließen oder umgekehrt. Damit ergibt sich

$$\mathrm{sim}(D_j, Q) = \frac{P(R|D_j)}{P(\overline{R}|D_j)} = \frac{P(D_j|R) \cdot P(R)}{P(D_j|\overline{R}) \cdot P(\overline{R})}.$$

Dabei ist

- $P(D_j|R) = P(N_Q(D_j)|R)$ die Wahrscheinlichkeit, dass ein zufällig gewähltes Dokument in R gerade aus $N_Q(D_j)$ ist,
- $P(R)$ die Wahrscheinlichkeit, dass ein zufällig gewähltes Dokument Q-elevant ist und

- $P(\overline{R})$ die Wahrscheinlichkeit, dass ein zufällig gewähltes Dokument Q-irrelevant ist.

Beim *Binary Independent Retrieval (BIR)* macht man folgende Grundannahmen:

- Ein Term kommt in einem Dokument entweder vor oder nicht (**b**inary). Verwandtheit von Termen gibt es nicht.
- Die Mengen N_i sind unabhängig (**i**ndependent).

Mit diesen Annahmen gilt:

$$\begin{aligned} P(D_j|R) &= P(N_Q(D_j)|R) \\ &= P\left(\left(\bigcap_{t_i\in\tau(Q)\cap\tau(D_j)} N_i\right) \cap \left(\bigcap_{t_i\in\tau(Q)\setminus\tau(D_j)} \overline{N_i}\right) \,\middle|\, R\right) \\ &\overset{\text{unabh.}}{=} \prod_{t_i\in\tau(Q)\cap\tau(D_j)} P(N_i|R) \cdot \prod_{t_i\in\tau(Q)\setminus\tau(D_j)} P(\overline{N_i}|R). \end{aligned}$$

Analog erhält man

$$\begin{aligned} P(D_j|\overline{R}) &= P(N_Q(D_j)|\overline{R}) \\ &= \prod_{t_i\in\tau(Q)\cap\tau(D_j)} P(N_i|\overline{R}) \cdot \prod_{t_i\in\tau(Q)\setminus\tau(D_j)} P(\overline{N_i}|\overline{R}). \end{aligned}$$

Zur Interpretation:

$r_i := P(N_i|R)$ ist die Wahrscheinlichkeit, dass der Term t_i in einem zufällig gewählten D_j, $j \in R$, vorkommt.

$n_i := P(N_i|\overline{R})$ ist die Wahrscheinlichkeit, dass der Term t_i in einem zufällig gewählten D_j, $j \in \overline{R}$, vorkommt.

Damit ist

$$\begin{aligned} \mathrm{sim}(D_j, Q) &= \frac{P(R)}{P(\overline{R})} \cdot \prod_{t_i\in\tau(Q)\cap\tau(D_j)} \frac{r_i}{n_i} \cdot \prod_{t_i\in\tau(Q)\setminus\tau(D_j)} \frac{1-r_i}{1-n_i} \\ &= \underbrace{\frac{P(R)}{P(\overline{R})}}_{\text{unabh. von } D_j} \cdot \prod_{t_i\in\tau(Q)\cap\tau(D_j)} \frac{r_i(1-n_i)}{n_i(1-r_i)} \cdot \underbrace{\prod_{t_i\in\tau(Q)} \frac{1-r_i}{1-n_i}}_{\text{nur abh. von } Q} \end{aligned}$$

Eigentlich interessiert uns nur ein Ranking der Dokumente und nicht die exakten Ähnlichkeitswerte. Daher lassen wir alle konstanten oder nur von Q abhängigen Faktoren weg und erhalten

$$\mathrm{sim}(D_j, Q) \sim \prod_{t_i\in\tau(Q)\cap\tau(D_j)} \frac{r_i(1-n_i)}{n_i(1-r_i)}.$$

Logarithmieren (streng monoton steigende Funktion, daher keine Änderung der Ordnung) liefert mit der abkürzenden Schreibweise

$$c_i := \log \frac{r_i(1-n_i)}{n_i(1-r_i)}$$

die Bewertungsfunktion

$$\mathrm{rsv}(D_j, Q) := \sum_{t_i \in \tau(Q) \cap \tau(D_j)} c_i.$$

1.9 Bemerkung Sind die c_i bekannt, so kann man die rsv-Werte schnell berechnen ($\rightarrow$ invertierte Listen, siehe später).

Soweit zur Analyse. Wir kommen nun auf die iterative Ermittlung der r_i und n_i zu sprechen:

Initialisierung:

Wir setzen für alle Terme $t_i \in \tau(Q)$

$$r_i := \frac{1}{2} \text{ und } n_i := \frac{\mathrm{df}_i}{N} = \frac{|N_i|}{N}.$$

Hier liegt die Heuristik zugrunde, dass Anfrageterme mit konstanter Wahrscheinlichkeit (= 0,5) in den relevanten Dokumenten und mit durchschnittlicher Wahrscheinlichkeit in den irrelevanten Dokumenten auftreten.

Feedback-Schritt:

Der Benutzer bestimmt aus den k präsentierten Dokumenten $D'_1, \ldots, D'_k$ die ℓ relevanten Dokumente $D'_{i_1}, \ldots, D'_{i_\ell}$. Wir setzen

$$k_i := \#\{j \mid t_i \in \tau(D'_j)\}$$

und

$$\ell_i := \#\{\lambda \mid t_i \in \tau(D'_{i_\lambda})\}.$$

ℓ_i bezeichnet die Anzahl der relevanten Dokumente (aus den k präsentierten Dokumenten), die t_i enthalten, k_i bezeichnet die Anzahl aller präsentierten Dokumente, die t_i enthalten.
In der nächsten Iteration setzen wir

$$r_i := \frac{\ell_i}{\ell} \text{ und } n_i := \frac{k_i - \ell_i}{k - \ell}.$$

Eine Variante zur Vermeidung von zu kleinen Werten ist

$$r_i := \frac{\ell_i + 0{,}5}{\ell + 1} \text{ und } n_i := \frac{k_i - \ell_i + 0{,}5}{k - \ell + 1}.$$

Vorteile des Binary Independent Retrievals:

- Dokumente werden mit absteigender Wahrscheinlichkeit ihrer Relevanz geordnet.
- Eine effiziente Auswertung ist möglich.

Nachteile des Binary Independent Retrieval:

- Die Initialisierung ist eine Heuristik.
- Termhäufigkeiten bleiben unberücksichtigt.
- Die angenommene Unabhängigkeit der Terme ist unrealistisch.

1.3 Termextraktion

Jetzt wenden wir uns der Frage zu, wie man von einem Dokument (in irgendeinem Format) an eine Menge indexierbarer Terme kommt. Hierzu kann man (grob) folgende Schritte verwenden:

1. Extraktion von Texten aus (strukturierten) Dokumenten, (Elimination sonstiger Struktur, wie Layout etc.)
2. Reduktion der Texte auf Folge „relevanter“ Wörter,
3. Strukturauswahl für Terme und Abbildung der Wortfolge auf eine Termfolge,
4. Reduktion von Wörtern oder Termen auf Stammformen,
5. Überführung von Termen in Indexterme (semantisch).

1.10 Bemerkung

- Der in Abschnitt 1.2 verwendete Termextraktor τ arbeitet auf dem Output von Schritt 1.
- τ kann einen oder mehrere der Schritte 2.-5. umfassen.
- In der Praxis werden oft nur einige der Schritte 1.-5. und dann auch nicht in exakt dieser Reihenfolge verwendet.

Die Termextraktion wird nun anhand eines Beispiels Schritt für Schritt vorgeführt.

1. Extraktion von Texten (aus strukturierten) Dokumenten

Institut für Informatik III
Universität Bonn

Homepage von Frank Kurth

Hochschulassistent im Projektbereich Kommunikationssysteme und Algorithmen, Arbeitsgruppe Prof. Dr. Michael Clausen.

Projekte und Interessen

- Multimedia Information Retrieval, insbesondere *Content-based Retrieval*
- Kommunikationsdienste und -schittstellen für multimediale Systeme
- Effiziente Algorithmen zur Verarbeitung multimedialer Signale

↓

Institut für Informatik III Universität Bonn Homepage von Frank Kurth Hochschulassistent im Projektbereich Kommunikationssysteme und Algorithmen Arbeitsgruppe Prof Dr Michael Clausen Projekte und Interessen Multimedia Information Retrieval insbesondere Content-based Retrieval Kommunikationsdienste und -schnittstellen für multimediale Systeme Effiziente Algorithmen zur Verarbeitung multimedialer Signale

2. Reduktion der Texte auf Folge „relevanter“ Wörter

Institut für Informatik III Universität Bonn Homepage von Frank Kurth Hochschulassistent im Projektbereich Kommunikationssysteme und Algorithmen, Arbeitsgruppe Prof Dr Michael Clausen Projekte und Interessen Multimedia Information Retrieval insbesondere Content-based Retrieval Kommunikationsdienste und schnittstellen für multimediale Systeme Effiziente Algorithmen zur Verarbeitung multimedialer Signale

↓

Institut ~~für~~ Informatik III Universität Bonn Homepage ~~von~~ Frank Kurth Hochschulassistent ~~im~~ Projektbereich Kommunikationssysteme ~~und~~ Algorithmen Arbeitsgruppe Prof Dr Michael Clausen Projekte ~~und~~ Interessen Multimedia Information Retrieval ~~insbesondere~~ Content-based Retrieval Kommunikationsdienste ~~und~~ schnittstellen ~~für~~ multimediale Systeme Effiziente Algorithmen ~~zur~~ Verarbeitung multimedialer Signale

3. Wahl Termstruktur und Abbildung der Wortfolge auf eine Termfolge

   Hier: Paare aus Termen mit Angabe der Position im Dokument

Institut Informatik III Universität Bonn Homepage Frank Kurth Hochschulassistent Projektbereich Kommunikationssysteme Algorithmen Arbeitsgruppe Prof Dr Michael Clausen Projekte Interessen Multimedia Information Retrieval Content-based Retrieval Kommunikationsdienste schnittstellen multimediale Systeme Effiziente Algorithmen Verarbeitung multimedialer Signale

(Institut; 0), (Informatik III; 1), (Universität; 2), (Bonn; 3), (Homepage; 4), (Frank; 5), (Kurth; 6), (Hochschulassistent; 7), (Projektbereich; 8), (Kommunikationssysteme; 9), (Algorithmen; 10), (Arbeitsgruppe; 11), (Prof Dr; 12), (Michael; 13), (Clausen; 14), (Projekte; 15), (Interessen; 16), (Multimedia; 17), (Information; 18), (Retrieval; 19, 21), (Content-based; 20), (Kommunikationsdienste; 22), (schnittstellen; 23), (multimediale; 24), (Systeme; 25), (Effiziente; 26), (Algorithmen; 27), (Verarbeitung; 28), (multimedialer; 29), (Signale; 30)

4. Reduktion von Wörtern oder Termen auf Stammformen

(Institut; 0), (Informatik III; 1), (Universität; 2), (Bonn; 3), (Homepage; 4), (Frank; 5), (Kurth; 6), (Hochschulassistent; 7), (Projektbereich; 8), (Kommunikationssysteme; 9), (Algorithmen; 10), (Arbeitsgruppe; 11), (Prof Dr; 12), (Michael; 13), (Clausen; 14), (Projekte; 15), (Interessen; 16), (Multimedia; 17), (Information; 18), (Retrieval; 19, 21), (Content-based; 20), (Kommunikationsdienste; 22), (schnittstellen; 23), (multimediale; 24), (Systeme; 25), (Effiziente; 26), (Algorithmen; 27), (Verarbeitung; 28), (multimedialer; 29), (Signale; 30)

(Institut; 0), (Informatik III; 1), (Universität; 2), (Bonn; 3), (Homepage; 4), (Frank; 5), (Kurth; 6), (*Hochschule*; 7), (*Assistent*; 7), (*Projekt*; 8,15), (*bereich*; 8), (*Kommunikation*; 9,22) (*system*; 9), (*Algorith*; 10,27), (*Arbeit*; 11), (*Gruppe*; 11), (Prof Dr; 12), (Michael; 13), (Clausen; 14), (*Interesse*; 16), (*Multimedia*; 17, 24, 29), (Information; 18), (Retrieval; 19, 21), (*Content*; 20), (*base*; 20), (*Dienst*; 22), (*Schnittstelle*; 23), (*System*; 25), (*Effizienz*; 26), (*verarbeiten*; 28), (*Signal*; 30)

Beachte insbesondere die Trennung von Komposita wie *Projektbereich*.

5. Überführung von Termen in Indexterme (semantisch)

(Institut; 0), (Informatik III; 1), (Universität; 2), (Bonn; 3), (Homepage; 4), (Frank; 5), (Kurth; 6), (Hochschule; 7), (Assistent; 7), (Projekt; 8,15), (bereich; 8), (Kommunikation; 9,22) (system; 9), (Algorith; 10,27), (Arbeit; 11), (Gruppe; 11), (Prof Dr; 12), (Michael; 13), (Clausen; 14), (Interesse; 16), (Multimedia; 17, 24, 29), (Information; 18), (Retrieval; 19, 21), (Content; 20), (base; 20), (Dienst; 22), (Schnittstelle; 23), (System; 25), (Effizienz; 26), (verarbeiten; 28), (Signal; 30)

↓

(Institut; 0), (Informatik III; 1), (*Universität, Hochschule*; 2, 7), (*Stadt:* Bonn; 3), (Homepage; 4), (*Name:* Frank; 5), (*Name:* Kurth; 6), (*Assistent, Gehilfe*; 7), (Projekt; 8,15), (*Bereich, Gebiet*; 8), (Kommunikation; 9,22) (system; 9), (Algorith; 10,27), (*Dienst, Arbeit*; 11, 22), (Gruppe; 11), (Prof Dr; 12), (*Name:* Michael; 13), (*Name:* Clausen; 14), (Interesse; 16), (Multimedia; 17, 24, 29), (Information; 18), (Retrieval; 19, 21), (Content; 20), (base; 20), (Schnittstelle; 23), (System; 25), (Effizienz; 26), (verarbeiten; 28), (Signal; 30)

Beachte Synonyme, z.B. *Bereich*, *Gebiet* (ähnliche Bedeutung).

1.3.1 Textextraktion

Ziel: Extraktion der Textteile eines Dokuments. Es geht dabei um die Textteile, die den *Inhalt* des Dokuments beschreiben (und nicht um Formatierungsanweisungen o.ä.).

- Schritt *vor* Termextraktor τ aus 1.2,
- *Output* ist etwa ein Textdokument wie in 1.2 modelliert.

Hier: Illustration anhand der Textextraktion aus einem HTML-Dokument. (Ähnliches gilt dann auch für Dokumente im PDF, Postscript, Word, etc. -Format)

1.11 Beispiel Umwandlung HTML → Textdokument

```
<HTML>

<HEAD>
   <TITLE>Homepage Frank Kurth</TITLE>
   <META> NAME="keywords"
          CONTENT="Homepage Informatik Clausen
                   Multimediaretrieval">
</HEAD>

<BODY>
<B>Projekte und Interessen</B>
<UL>
<LI>Multimedia Information Retrieval, insbesondere
    <I>Content-based Retrieval</I></LI>
<LI>Kommunikationsdienste und -schnittstellen
    f&uuml;r multimediale Systeme</LI>
<LI>Effiziente Algorithmen zur Verarbeitung
    multimedialer Signale</LI>
...
```

Extrahiert wird:

```
Homepage Frank Kurth
Homepage Informatik Clausen Multimediaretrieval Projekte
und Interessen Multimedia Information Retrieval, insbesondere
Content-based Retrieval Kommunikationsdienste und
-schnittstellen für multimediale Systeme Effiziente
Algorithmen zur Verarbeitung multimedialer Signale
```

Termextraktionsschritte hier:

1. Konkatenation der Inhalte bestimmter Tags, z.B. `<Titel>`, `<BODY>`, Content-Attribut des `<Meta>`-Tags
2. Elimination verbliebener Tags, z.B. `<H1>, <B>, <I>`, etc.
3. Umwandlung von Sonderzeichen in Zeichen eines geeigneten Alphabets, z.B.

   ` ` $\mapsto \sqcup$ oder `&auml;` $\mapsto$ ä

1.12 Bemerkung Bei anderen Retrievalarten als dem reinen Textretrieval können weitere Informationen aus der Struktur extrahiert werden, z.B. Daten des Zieldokuments bei Hyperlinks (siehe §2 Webretrieval).

1.3.2 Reduktion auf relevante Wörter

Ideen zur Wortrelevanz:

1. Wörter mit geringer Aussagekraft tragen nicht viel zur Diskrimination von Dokumenten bei und können weggelassen werden. Beispiele für Wörter mit geringer Aussagekraft sind `und, genau, wird, über, es,...`
2. Sehr selten auftretende Wörter (selten sowohl bzgl. Dokumentenkollektion als auch bzgl. Anfrage) können weggelassen werden, wenn deren Vorhandensein den durch sie entstehenden Effizienzverlust (Speicherplatz, Dauer Anfragebearbeitung) nicht aufwiegt. Ein Beispiel ist das Wort
   `Humuhumu-nukunuku-a-pua'a` (Staatsfisch von Hawai'i)

1.13 Bemerkung

- Punkt 2. ist eher für spezialisierte Datenbestände gedacht: Für allgemeine Retrievalaufgaben (z.B. WWW-basiert) erwartet der Benutzer auch Treffer für exotische Anfragen.

  (Im letzten Beispiel liefert z.B. Google einen Treffer für den Hawai'i-Fisch, wenn man nur `Humuhumu` anfragt.)

- Benutzer-Feedback kann hier einbezogen werden. Modell: Häufig angefragte Terme sind aus Benutzersicht „relevant“.

Qualitative Betrachtung der Relevanz von Termen für die Indexierung
Betrachten nun

- Dokumentenkollektion $\mathcal{D} = (D_1, \ldots, D_M)$,
- $\tau : L \to T \sqcup \{\uparrow\}$ Termextraktor wie in 1.2., den man entsprechend auf die Dokumente D_i und die Kollektion $\mathcal{D}$ zu einem Termextraktor $\tau_{\mathcal{D}}$ fortsetzt.

Für Term $t_i \in T$ sei

$$N_i := |\tau_{\mathcal{D}}^{-1}(t_i)|$$

die Anzahl aller Auftreten des Terms in $\mathcal{D}$ inklusive Vielfachheiten, sowie

$$N := N_1 + \ldots + N_{|T|}$$

die Anzahl aller Wortvorkommen. Die Terme seien o.B.d.A. absteigend nummeriert,

$$N_1 \geq \ldots \geq N_{|T|}.$$

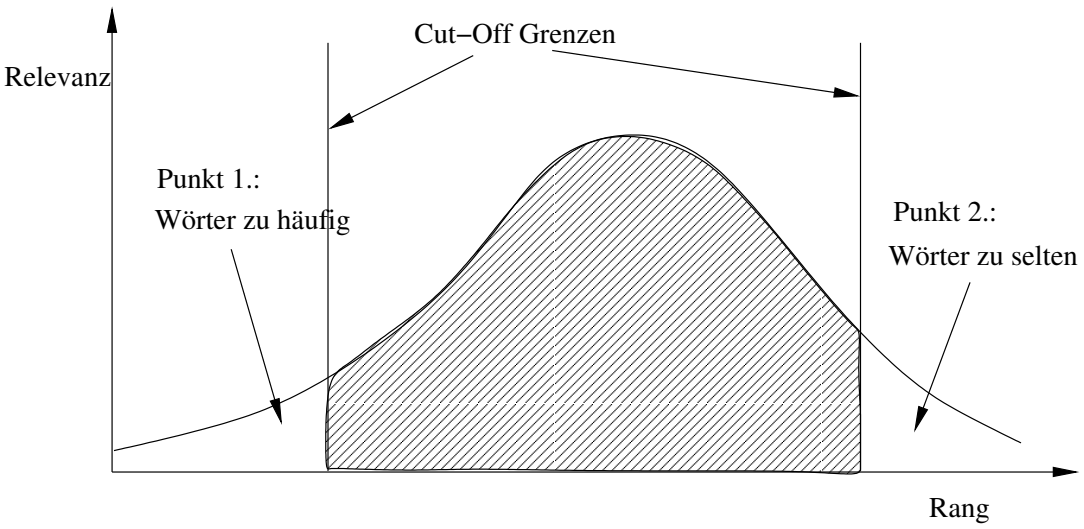
Wir sagen, Term t_i hat *Rang* i. Dann ist

$$p_i := \frac{N_i}{N}$$

der Anteil an Termen vom Rang i an allen Wortvorkommen.
(Natürlich ist

$$\sum_{i=1}^{|T|} \frac{N_i}{N} = 1,$$

d.h. $i \mapsto p_i$ ist W-Verteilung.)
Obige Überlegungen 1. und 2. legen folgende qualitative Sicht der *Relevanz* von Termen *abhängig von ihrem Rang* nahe:



Wir wollen wissen: Wie sieht die Verteilung $i \mapsto p_i$ ungefähr aus?

Anwendung:

- Festlegen von Cut-Off Schranken für obige Kriterien 1. und 2.
- Bestimmung des ungefähren Rangs bei gegebener Häufigkeit p_i

Mögliche **Vorgehen** zur Bestimmung der Verteilung:

1. Statistische Ermittlung von $i \mapsto p_i$ durch Betrachtung großer Dokumentensammlungen und Hochrechnung (Übung: Wie kann man hier vorgehen?).
2. Bestimmung einiger p_i's plus geeignete Modellierung der restlichen Werte der Verteilung.

Es stellt sich heraus, dass viele Verteilungen in der Praxis (Bevölkerungszahlen von Städten, Webseitenzugriffe,...) durch das sogenannte *Gesetz von Zipf* modelliert werden können.
Der Ansatz besteht darin, die Verteilung der p_i durch eine hyperbolische Funktion p mit einer geeigneten Konstante c zu modellieren:

$$p(i) := \frac{c}{i}.$$

Wir erhalten

$$c = p(i) \cdot i,$$

in unserem Anwendungsfall also

$$\text{(relative Termhäufigkeit)} \cdot \text{Rang} = \text{konst.}$$

Approximiert man $p(i) \approx p_i$, erhält man

$$\frac{N_i}{N} \approx \frac{c}{i}, \text{ also } i \approx \frac{cN}{N_i},$$

d.h. ein Term, der N_i-mal vorkommt, hat ungefähr Rang i.

Wie sieht die Konstante c aus, falls tatsächlich $i \mapsto p(i) \equiv i \mapsto p_i$?

Dann ist

$$1 = \sum_{i=1}^{|T|} p_i = \sum_{i=1}^{|T|} \frac{c}{i} \Leftrightarrow c = \frac{1}{\sum_{i=1}^{|T|} \frac{1}{i}} = \frac{1}{\ln |T| + 0{,}5772 + O(1/|T|)},$$

da $H_n := \sum_{k=1}^{n} \frac{1}{k} = \ln n + 0,577\ldots + O(1/n)$. (Beachte: c ist abhängig von $|T|$.)

Im doppelt logarithmischen Koordinatensystem hat diese Modellierung eine anschauliche Interpretation:

Logarithmieren von $p(i) = \frac{c}{i}$ liefert

$$\ln p(i) = \ln c - \ln i.$$

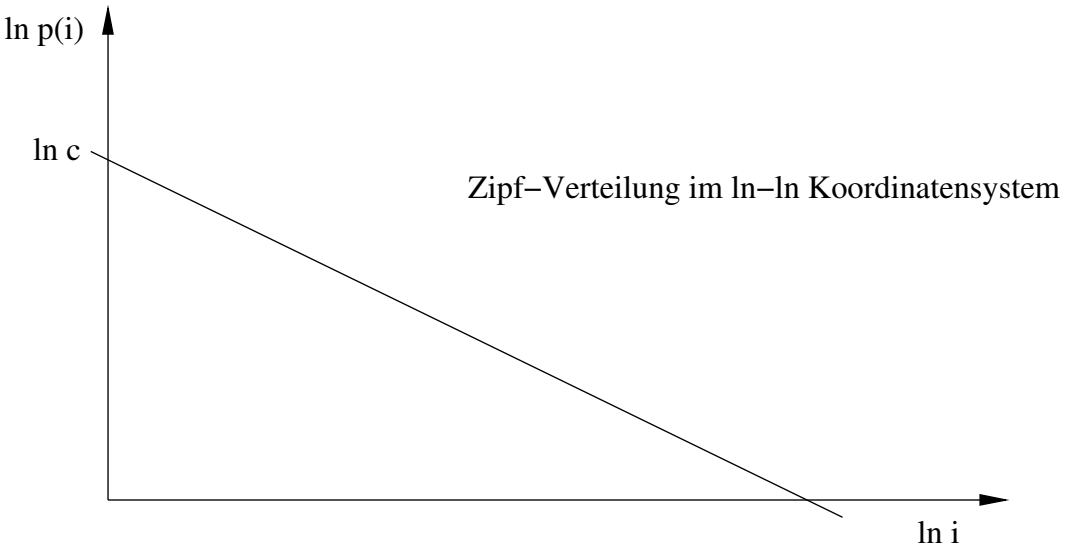
Trägt man nun $\ln p(i)$ gegen $\ln i$ auf — dies entspricht der Funktion

$$\ln i \mapsto \ln p(i) = -\ln i + \ln c,$$

so erhält man die Funktion

$$P : x \mapsto -x + \ln c,$$

also eine Gerade



der Steigung $\frac{\mathrm{d}}{\mathrm{d}x}P \equiv -1$.

1.14 Bemerkung

- Allgemeiner kann man noch

  $$p_\theta : i \mapsto \frac{c}{i^\theta}$$

  als Modellierungsansatz wählen. In ln-ln-Koordinaten erhöht dies den Freiheitsgrad um eine beliebig wählbare Steigung θ der Geraden (Übung).

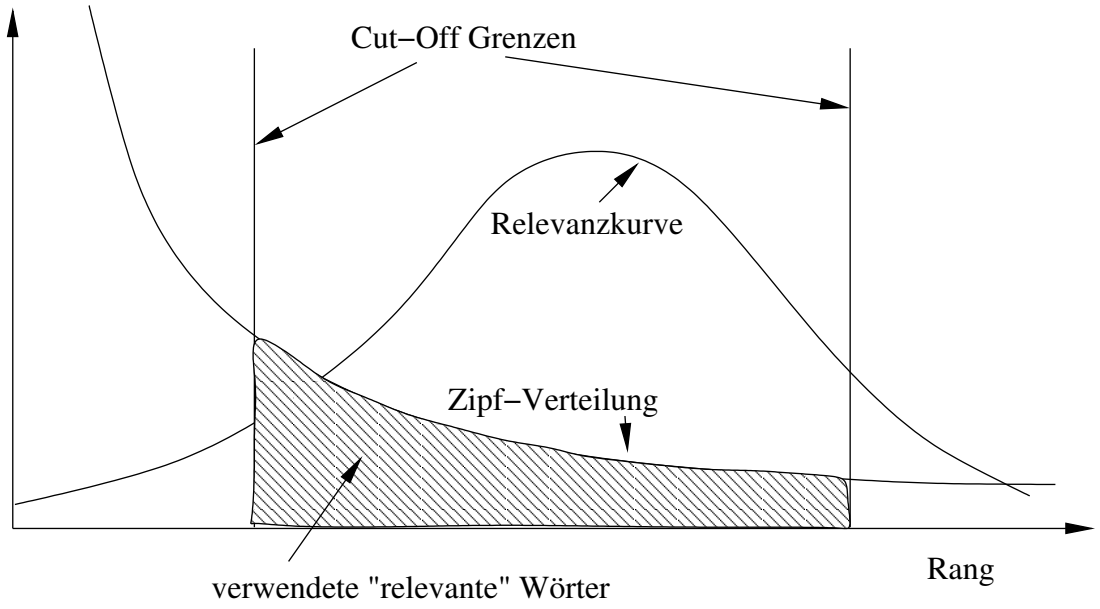
- Sehr gebräuchlich ist auch das sog. *2. Gesetz von Zipf*, das grob die Form

  $$\#\text{ Terme, die } i\text{-mal vorkommen } = \frac{\text{const.}}{i}$$

  besitzt (auch *Pareto*-Verteilung genannt).

  Übung: Wo werden solche Verteilungen verwendet? Wie sieht eine typische Pareto-Verteilung (im Logarithmischen) aus?

Qualitativ: Wortverteilung und Relevanz versus Rang in unserer Anwendung:



Noch ein paar **Statistiken**:

- *Stopwörter* ≡ Wörter mit geringer Aussagekraft machen ca. 20-30% aller Wörter aus.
- Auch die 200–300 häufigsten Wörter einer Dokumentenkollektion haben oft wenig Aussagekraft.

1.3.3 Auswahl der Termstruktur und Abbildung auf Termmenge

Hier:

1. Modellierung des zu indexierenden Termtyps,
2. Entscheidung, wieviel Struktur des Dokuments in den Index übernommen wird.

Zu 1.: Gegeben ist ein Dokument, etwa in Form von Wortfolge aus L^+. Mögliche Termtypen:

- Teilmengen $T \subset L$ der Sprache, z.B. unter Weglassen von Großschreibung und/oder Fehlerkorrektur ($\rightarrow$ Anfrage).

- N-Gramme: Sei $w = w_1 \dots w_n \in L \cap \Sigma^n, n \geq N$ ein Wort der Länge n aus L, dann wird w die N-Gramm-Folge

$$w_1 \dots w_N, w_2 \dots w_{N+1}, \dots, w_{n-N+1} \dots w_n$$

zugeordnet. Beispiel: $N = 3$.

$$\text{Clausen} \mapsto \text{Cla, lau, aus, use, sen}$$

(Ist $n < N$ kann man mit Leerzeichen auffüllen.)

Vorteile: Keine Stammformreduktion mehr nötig; einzelne Tippfehler wiegen nicht so schwer.

Zu 2.: Sei t Term aus Termmenge T, τ Termextraktor, sowie $\mathcal{D} = (D_1, \dots, D_M)$, mit Dokumenten $D_i = (d_{i1}, \dots, d_{ik_i})$, Dokumentenkollektion mit Termextraktor τ wie oben.
Strukturparameter, die indexiert werden können:

▷ Sind für $i \in [1 : M]$ Dokumente D_i gegeben, so können weiterhin alle *Wortpositionen*

$$L_{\mathcal{D}}(t, i) = \{\ell \mid \tau(d_{i\ell}) = t\}$$

des Terms t im Dokument i indexiert werden. Manchmal speichert man die Dateinummer mit ab:

$$L_{\mathcal{D}}(t) := \{(i, \ell) \mid \tau(d_{i\ell}) = t\}.$$

1.15 Bemerkung Später werden wir die Termextraktion nicht immer explizit behandeln. Für eine Dokumentenkollektion $\mathcal{D} = (D_1, \dots, D_M)$ über L und $t \in L$ verwenden wir dann auch die Schreibweise

$$L_{\mathcal{D}}(t) := \{(i, \ell) \mid d_{i\ell} = t\}.$$

Dies führt dann zu sogenannten *invertierten Listen.*

▷ $\text{tf}(t, i) := |L_{\mathcal{D}}(t, i)|$, Anzahl Vorkommen des Terms t im Dokument D_i oder *Termhäufigkeit* von t in D_i (tf ≡ *Term frequency*).

1.3.4 Stammformreduktion (engl. Stemming)

Das Ziel des Stemmings ist die Reduktion verschiedener Formen eines Wortes auf eine einheitliche Form. Verschiedene Wortformen entstehen z.B. durch

▷ Singular/Plural ▷ Zeitformen
▷ Deklination ▷ Konjugation
▷ Konjunktiv ▷ ...

Beispiel:

$$\left.\begin{array}{c}\text{sitzen}\\ \text{gesessen}\\ \text{saß}\\ \text{sitzend}\end{array}\right\} \quad \mapsto \quad \text{sitzen}$$

Vorgehen zur Stammformbildung:

1. Verwendung von *Termumformungsregeln*

   Vorteil: Kompakt realisierbar. Nachteil: Nur approximatives Stemming, grammatikalisch fehlerhaft.

2. Benutzung eines *Wörterbuchs*

   Vorteil: Exaktes Stemming. Nachteil: Speicherplatzbedarf für Wörterbuch und Zeitbedarf zum Nachschlagen eines Wortes.

Stemming im Deutschen:

Variante 1. kaum möglich, da zu viele Sonderfälle — unterschiedliche Wörter ändern sich auf zu unterschiedliche Arten.
Beispiel:

fliegen	sitzen	gehen	stellen
geflogen	gesessen	gegangen	gestellt

Weiterhin: Zusammengesetzte Wörter wie `Donaudampfschiffahrtsgesellschaft` treten auf (üblich: Aufspalten dieser Wörter).

Daher: Verwenden der Wörterbuchvariante.

Stemming im Englischen:

Hier gibt es eine Lösung mit Termumformungsregeln, den *Porter Algorithmus* [1]. Beschreiben hier Grobablauf des Porter Algorithmus mit einigen exemplarischen Termumformungsregeln.

Vorbereitung:
Sei $\Sigma := \{A, \ldots, Z\}$ das Alphabet (Vereinfachung: nur Großbuchstaben!). Ein Wort $w = w_1 \ldots w_n \in \Sigma^+$ wird nun wie folgt in eine *Vokal-Konsonanten-Form* $w' \in \{v, c\}^n$ gebracht:

1. $$w'_i := \begin{cases} \text{v} & \text{falls } w_i \in \{A, E, I, O, U\} \\ & \text{oder } (i > 1, w_i = \text{Y und } w_{i-1} \notin \{A, E, I, O, U\}), \\ \text{c} & \text{sonst.} \end{cases}$$

   Beispiel: GREEDY $\mapsto$ ccvvcv

2. Man kann das Wort $w' \in \{\mathrm{c,v}\}^+$ eindeutig in der Form

$$w' = \mathrm{c}^*(\mathrm{v}^+\mathrm{c}^+)^m\mathrm{v}^*$$

mit einem $m \geq 0$ schreiben. Hierbei bedeutet $(\mathrm{v}^+\mathrm{c}^+)^m$ das m-malige Vorkommen von Teilwörtern vom Typ $\mathrm{v}^+\mathrm{c}^+$:

$$(\mathrm{v}^+\mathrm{c}^+)^m \equiv \underbrace{\mathrm{v}^+\mathrm{c}^+ \ldots \mathrm{v}^+\mathrm{c}^+}_{m-\text{mal}}.$$

Beispiel: ccvvcv $= \mathrm{c}^2(\mathrm{v}^2\mathrm{c})\mathrm{v}$

Porter Algorithmus

Nun werden folgende Schritte 1–5 nacheinander ausgeführt.

- Input ist Wort $w \in \Sigma^+$.
- Fallunterscheidung nach passenden Suffixen y von w. Schreiben $w = zy$.
- Eindeutige Vokal-Konsonanten-Form z' des Präfixes z legt Zusatzbedingungen fest.
- $T_i(y)$ beschreibt dann die Suffixumformung in Schritt i.

	Regel			**Beispiel**	
Schritt	y	z'	$T_i(y)$	w	$T_i(w)$
1 a)	SSES		SS	CARESSES	CARESS
	IES		I	PONIES	PONI
	SS		SS	CARESS	CARESS
	S		ε	CATS	CAT
b)	EED	$m > 0$	EE	FEED	FEED
	ED	$^*\mathrm{v}^*$	ε	PLASTERED	PLASTER
	ING	$^*\mathrm{v}^*$	ε	MOTORING	MOTOR
2	ATIONAL	$m > 0$	ATE	RELATIONAL	RELATE
	TIONAL	$m > 0$	TION	CONDITIONAL	CONDITION
	ENCI	$m > 0$	ENCE	VALENCI	VALENCE
	IZER	$m > 0$	IZE	DIGITIZER	DIGITIZE
	etc.				
3	ICATE	$m > 0$	IC	TRIPLICATE	TRIPLIC
	ATIVE	$m > 0$	ε	FORMATIVE	FORM
	ALIZE	$m > 0$	AL	FORMALIZE	FORMAL
4	TION	$m > 1$	T	ADOPTION	ADOPT

	SION	$m > 1$	S	COMPREHENSION	COMPREHENS
	OU	$m > 1$	ε	HOMOLOGOU	HOMOLOG
	ISM	$m > 1$	ε	PLATONISM	PLATON
	etc.				
5 a)	E	$m > 1$	ε	RATE	RATE
	E	$m = 1 \wedge (1)$	ε	CEASE	CEAS
	L	$m > 1 \wedge (2)$	ε	CONTROLL	CONTROL

Hierbei bedeuten:

- ▷ *v*, dass mindestens ein v in der Zerlegung enthalten ist.
- ▷ (1): $z' = z''\mathrm{c}^+\mathrm{v}^+\mathrm{c}^+$ und $z = z_1 \ldots z_\ell$, wo $z_\ell \notin \{\mathrm{W}, \mathrm{X}, \mathrm{Y}\}$.
  Beispiele: $z = x\mathrm{WIL}$, $z = x\mathrm{HOP}$.
- ▷ (2): $z' = z''\mathrm{cc}$ und $z = z_1 \ldots z_k$, wo $z_{k-1} = z_k$
  (Doppelter Konsonant am Ende des Wortes).

1.3.5 Semantische Abbildung auf Indexterme

1. Behandlung von Mehrfach- und Gleichbedeutung von Worten

- *Homonyme:* Ein Wort hat mehrere Bedeutungen (→ Kontext!)
  Beispiel:

  „streifen“ im Sinne von $\begin{cases} \text{„umher streifen“} \\ \text{„den Radfahrer streifen“} \end{cases}$

- *Synonyme:* Worte mit der selben oder ähnlicher Semantik
  Beispiel: „gehen“ und „laufen“
  Achtung: Manchmal ist Wort Homonym und Synonym anderer Worte, z.B. werden oft auch „laufen“ und „rennen“ synonym gebraucht.

Auswege: Einbeziehung von Kontext (Homonyme), Thesauri (Synonyme, Homonyme)

2. Kontextabhängige Semantik

Worte haben manchmal nur im Kontext mit anderen Worten eine diskriminierende Semantik, wie zum Beispiel im Falle des Zitats „To be or not to be“. Um auch hier eine Indexierung zu erreichen, verwendet man eine

- Indexierung ganzer *Phrasen* (≡ Wortfolgen)
- oder benutzt N-Gramme.

1.4 Evaluation von IR-Systemen

Bei der Bewertung von IR-Systemen können u.a. folgende Daten betrachtet werden:

- Relevanz der (Treffer der) Treffermenge
- Fehlertoleranz
- Benutzerfreundlichkeit
- Effizienz

1.16 Bemerkung

▷ Bei Datenbanken ist die Treffermenge systemunabhängig (hängt nur vom Datenmodell ab). Hier liefert Relevanzbewertung der Treffermenge also *keine* Aussage über das DB-System. (Datenretrieval vs. Information-Retrieval)

▷ *Relevanz* muss vom Benutzer bewertet werden (vgl. Relevanz-Feedback beim probabilistischen Retrieval).

1.4.1 Testkollektionen

Als Referenz werden Testkollektionen verwendet. Diese stellen standardisierte

- (große) Dokumentenkollektionen,
- Anfragen und
- relevante Dokumente für alle Anfragen

zur Verfügung.

1.17 Beispiel

▷ TREC-Kollektion (Text Retrieval Conference): umfangreiches Testmaterial im Bereich Textretrieval, siehe `http://trec.nist.gov/`

▷ Datensatz Reuters 21578, von gleichnamiger Nachrichtenagentur zur Verfügung gestellt (Wert von textuellen Nachrichtendaten nimmt kurz nach deren Veröffentlichung stark ab!). Für Infos zu neuen Testdatensätzen von Reuters siehe

`http://about.reuters.com/researchandstandards/corpus/index.asp`

▷ Es existieren einige (auch spezialisierte) Bilddatenbanken für den Bereich Bildretrieval.

▷ Eine TREC-Kollektion existiert auch im Video-Bereich (TREC-Video).

▷ Momentan gibt es Bestrebungen im Bereich des Musik-IR Testkollektionen und entsprechende Evaluationswettbewerbe für Retrievalalgorithmen zu etablieren. Wichtigstes Beispiel ist die internationale MIREX-Initiative, siehe `http://www.music-ir.org/evaluation/`.

1.4.2 Precision, Recall und Fallout

Seien

- $\mathcal{D} = (D_1, \ldots, D_M)$ Dokumentenkollektion und $I = [1 : M]$ die Menge der Dokumenten-IDs,
- q eine beliebige aber fest gewählte (und hier nicht näher spezifizierte) Anfrage,
- $R_q \subseteq I$ die Menge aller zu q relevanten Dokumente,
- sowie $T_q \subseteq I$ die Treffermenge des zu evaluierenden Systems zur Anfrage q.

Definiere:

▷ die *Precision* des Systems bzgl. q als

$$p_q := \frac{|T_q \cap R_q|}{|T_q|},$$

den *Anteil der relevanten Dokumente an der Treffermenge.*

▷ den *Recall* des Systems bzgl. q als

$$r_q := \frac{|T_q \cap R_q|}{|R_q|},$$

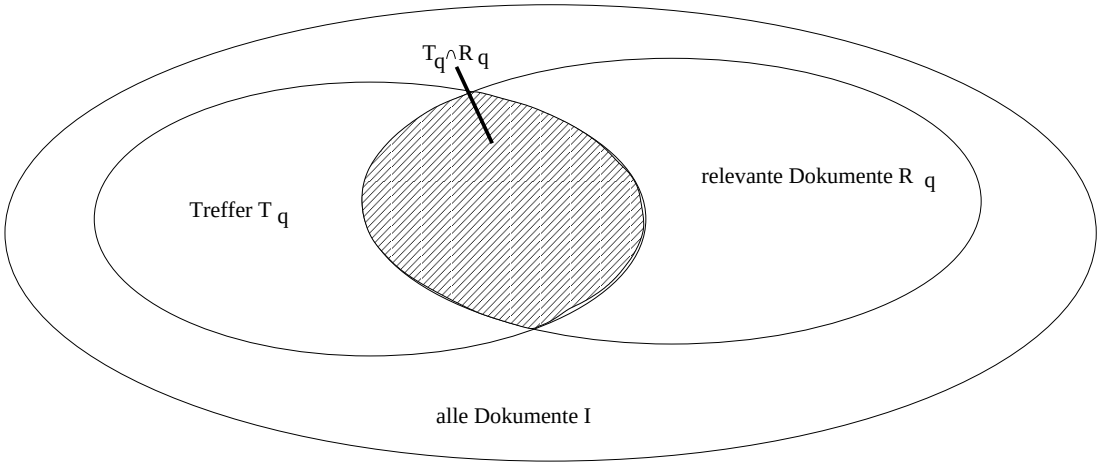
den *Anteil der Treffermenge an den relevanten Dokumenten.*

▷ den *Fallout* des Systems bzgl. q als

$$f_q := \frac{|T_q \backslash R_q|}{|I \backslash R_q|}$$

das *Verhältnis nicht relevanter Treffer zu allen nicht relevanten Dokumenten.*

Illustration:



Üblich bei der Evaluation ist auch die Mittelung über die Ergebnisse von n Anfragen $q_1, \ldots, q_n$, die sogenannte *Makrobewertung*

$$\overline{p} := \frac{1}{n} \sum_{i=1}^{n} p_{q_i} \qquad \text{und} \qquad \overline{r} := \frac{1}{n} \sum_{i=1}^{n} r_{q_i}.$$

Alternativ: *Mikrobewertung*

$$\tilde{p} := \frac{\sum_{i=1}^{n} |T_{q_i} \cap R_{q_i}|}{\sum_{i=1}^{n} |T_{q_i}|} \qquad \text{und} \qquad \tilde{r} := \frac{\sum_{i=1}^{n} |T_{q_i} \cap R_{q_i}|}{\sum_{i=1}^{n} |R_{q_i}|}.$$

W-Theoretische Interpretation von Precision und Recall

Auf I sei eine W-Verteilung P gegeben. Dann ist

$$P(R_q|T_q) = \frac{P(R_q \cap T_q)}{P(T_q)}.$$

Bei Gleichverteilung ist dies

$$= \frac{|R_q \cap T_q|}{|T_q|} = p_q.$$

Analog erhält man $P(T_q|R_q) = r_q$. Mit dem Satz von Bayes erhalten wir

$$P(R_q|T_q) = P(T_q|R_q)\frac{P(R_q)}{P(T_q)},$$

bei Gleichverteilung insbesondere

$$r_q = p_q \frac{|T_q|}{|R_q|}.$$

Anwendung dieser allgemeineren Sichtweise: Bewertung von Rankings.

Sei $P : I \to [0,1]$ eine W-Verteilung auf allen Dokumentennummern, die die Treffer aus T_q absteigend gemäß dem Ranking des IR-Systems gewichtet (und etwa $P(i) = 0$ falls $i \notin T_q$).
Dann „berücksichtigt“ die verallgemeinerte Precision $P(R_q|T_q)$ dieses Ranking.
Interpretation: Der Benutzer betrachtet meist sowieso nur die obersten (ersten) Treffer, die das System liefert. Darum werden diese stärker bewertet als die weiteren Treffer.

1.4.3 Precision-Recall Diagramme

Eine etablierte Möglichkeit, Retrievalergebnisse zu beurteilen, denen ein Ranking zugrunde liegt, sind Precision-Recall (PR-) Diagramme:
Sei $T_q \subseteq I$ Treffermenge zur Anfrage q, sowie $r : T_q \to [0,1]$ ein Ranking der Treffermenge. Sei

$$d_1, \ldots, d_{|T_q|}$$

eine Sortierung der Treffer in absteigender Reihenfolge des Rankings.
Interpretation: Dokument D_{d_i} hat höheren Rang als D_{d_j} (genauer: $r(d_i) \geq r(d_j)$) gdw. $i < j$. Beachte, dass die Sortierung, falls mehrere Dokumente den gleichen Rankingwert besitzen, nicht eindeutig ist. In diesem Fall wählen wir zufällig eine passende Sortierung. Eigentlich sollte jedoch der Retrievalalgorithmus für eine eindeutige Sortierung (bzw. verschiedene Rankingwerte) sorgen. Haben mehrere Dokumente den gleichen Rang, würde man erwarten (Idealfall), dass sie entweder alle relevant oder alle nicht relevant sind.
Definiere für $i \in [1 : |T_q|]$ die Menge

$$T_{q,i} := \{d_1, \ldots, d_i\}$$

der ersten i Treffer in obiger (Rang-) Reihenfolge und bestimme jeweils

$$p_{q,i} := \frac{|T_{q,i} \cap R_q|}{|T_{q,i}|} \qquad \text{und} \qquad r_{q,i} := \frac{|T_{q,i} \cap R_q|}{|R_q|},$$

dann heißt die Menge

$$\{(r_{q,i}, p_{q,i}) \mid i \in [1 : |T|]\}$$

Precision-Recall-Diagramm des Systems zur Anfrage q.

1.18 Beispiel Die Abbildung 1.3 zeigt das Precision-Recall Diagramm für die beispielhaften Precision-Recall Werte aus Tabelle 1.2. Meist wählt man zur Darstellung des Precision-Recall-Diagramms den Übergang zu einer interpolierten Version wie in Abbildung 1.4.

1.19 Bemerkung (Auch zum vorhergehenden Beispiel)

- Wünschenswert: $p_{q,\cdot} \equiv 1$ (alle Treffer sind relevant).

i	d_i	$d_i \stackrel{?}{\in} R_q$	$p_{q,i}$	$r_{q,i}$
1	588	$\surd$	1.00	0.2
2	589	$\surd$	1.00	0.4
3	576		0.67	0.4
4	590	$\surd$	0.75	0.6
5	986		0.6	0.6
6	592	$\surd$	0.67	0.8
7	984		0.57	0.8
8	988		0.50	0.8
9	578		0.44	0.8
10	985		0.40	0.8
11	103		0.36	0.8
12	591		0.33	0.8
13	772	$\surd$	0.38	1.0
14	990		0.36	1.0

Abbildung 1.2: Beispiel für Precision-Recall Werte.

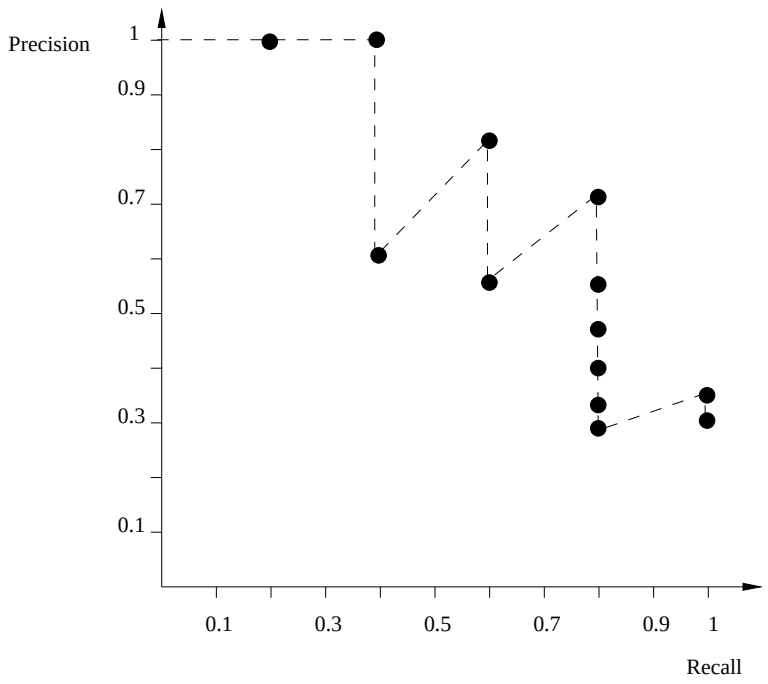


Abbildung 1.3: Precision-Recall Diagramm für die Werte aus Tabelle 1.2.

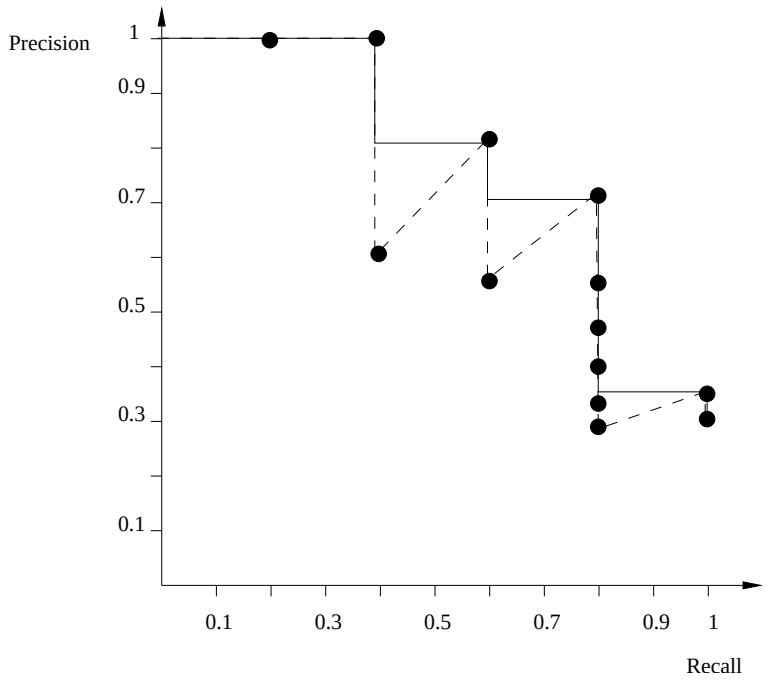


Abbildung 1.4: Precision-Recall Diagramm für die Werte aus Tabelle 1.2.

- Grob: Das Ansteigen des Recalls entspricht dem Auffinden eines weiteren *relevanten* Dokuments. Nimmt man weitere *nicht relevante* dazu, sinkt die Precision bei gleichem Recall.
- Man kann den mittleren Abstand des Graphen zum Punkt $(1, 1)$ als Gütemaß verwenden.
- Auch „gemittelte“ PR-Diagramme sind möglich.

  Übung: Diskutiere verschiedene Möglichkeiten hierzu!

1.4.4 Weitere Evaluationsmaße (stichwortartig)

- Kombinierte Maße:
  - *Harmonisches Mittel:*
    $$H_q := \frac{2}{\frac{1}{r_q} + \frac{1}{p_q}}$$
    (Hoher Wert von H_q nur, wenn Precision p_q und Recall r_q hoch sind. Annahme: PR-Werte alle in $[0, 1]$.)
  - *E-Maß:*
    $$E_q := 1 - \frac{1 + b^2}{\frac{b^2}{r_q} + \frac{1}{p_q}},$$

wo $b > 0$ gewichtet, ob entweder die Precision ($b > 1$) oder der Recall ($b < 1$) stärker gewichtet werden soll.

Beide Maße erweitert man entsprechend unter Verwendung der PR-Folgen $(p_{q,i})_i$ und $(r_{q,i})_i$ des letzten Abschnitts, s.d. man Kurven zur Bewertung eines Rankings erhält.

- *Nützlichkeitsmaß*, Idee: Verwende Benutzerfeedback.

  Benutzer darf zu je zwei Dokumenten bewerten, welches er präferiert (welches für ihn relevanter ist). Man testet dann das Ranking eines IR Systems, indem man

  – Übereinstimmungen mit der Benutzerpräferenz (positiv) und
  – Gegensätzliche Präferenzen (negativ)

  zählt. Dieses Verfahren wiederholt man für mehrere Anfragen. So ist auch Vergleich zweier IR-Systeme möglich.

- *Coverage* und *Novelty*. Sei $U_q \subseteq R_q$ die Menge relevanter Dokumente, die dem Benutzer bereits bekannt waren. Die *Coverage* ("Abdeckung")

$$c_q := \frac{|T_q \cap U_q|}{|U_q|}$$

  ist der Anteil an den bereits bekannten Dokumenten, die zur Treffermenge gehören.

  Die *Novelty*

$$n_q := \frac{|(T_q \cap R_q) \backslash U_q|}{|T_q \cap R_q|}$$

  ist der Anteil der relevanten, dem Benutzer vorher unbekannten Treffer an allen relevanten Treffern.

Für weiteres zu Evaluationsmaßen verweisen wir auf die Literatur (z.B. das *Modern IR*-Buch [2]).

1.5 Indexstrukturen und Retrievalalgorithmen

Prinzip: Vorverarbeitung des Datenbestands und Erstellung einer Suchstruktur (*Index*), die eine effiziente Anfragebearbeitung erlaubt.

Unterscheiden Indexe für

- zeichenkettenbasiertes Retrieval und
- termbasiertes Retrieval.

Bemerkung: Abgrenzung zu Suchstrukturen für herkömmliche schlüsselbasierte Suche (Bäume, Hashing,...) aufgrund der Anforderungen

- strukturierte Anfragen (z.B. boolesche Ausdrücke) mit mehreren Anfragetermen sowie
- unscharfe Suche (Fehlertoleranz, Ähnlichkeitssuche) zu ermöglichen.

1.5.1 Zeichenkettenbasiertes Retrieval

Bei verschiedenen Stringmatching-Algorithmen wird der *Anfragestring* q, $|q| =: m$, vorverarbeitet.

1.20 Beispiel

- ▷ Konstruktion eines (minimalen) endlichen Automaten, der $\Sigma^* q$ akzeptiert,
- ▷ Skip-Tabellen bei den Algorithmen nach Knuth-Morris-Pratt oder Boyer-Moore.

So erreicht man — wenn n die Länge des zu durchsuchenden Texts ist — Laufzeiten von $O(n+m)$ oder sogar $O(n)$ (Komplexitätsmaß beruht auf Zeichenvergleichen).
Hier: Vorverarbeitung des *zu durchsuchenden* Strings.
Dies führt zu Laufzeiten von $O(m)$ Zeichenvergleichen zur Entscheidung des Substringproblems $q \overset{?}{<_s} x$, wobei

$$q <_s x :\Leftrightarrow \exists v, w \in \Sigma^* : vqw = x.$$

Wir skizzieren hier grob einige Verfahren und Datenstrukturen (weitere Details, siehe Übungen).

1.5.1.1 Tries (von Retrieval & Tree)

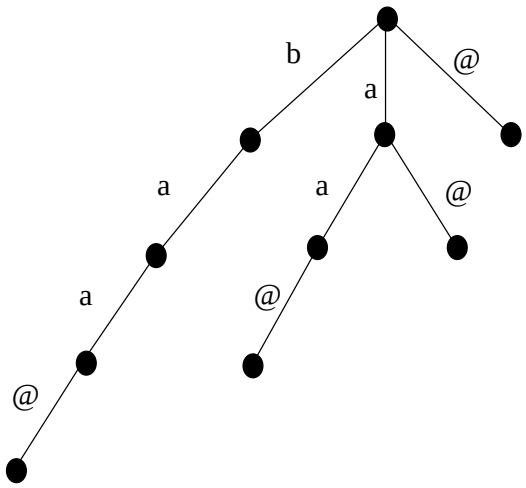
Betrachten Textdokumente der Form $w \in \Sigma^*@$ wo $@ \notin \Sigma$ Endezeichen ist. Trie(w) ist ein Baum (V, E) mit folgenden Eigenschaften:

1. Die Kanten sind mit Zeichen aus $\Sigma \cup \{@\}$ beschriftet und jedes Zeichen kommt in den ausgehenden Kanten eines Knotens höchstens einmal vor.
2. Sei $\ell(v)$ die Kantenbeschriftung entlang eines Pfades von der Wurzel zum Blatt $v \in V$, dann ist

   $$\{\ell(v) \mid v \in V \text{ ist Blatt}\}$$

   die Menge der nichtleeren Suffixe von w.

1.21 Beispiel $w = baa@$, Trie(w)



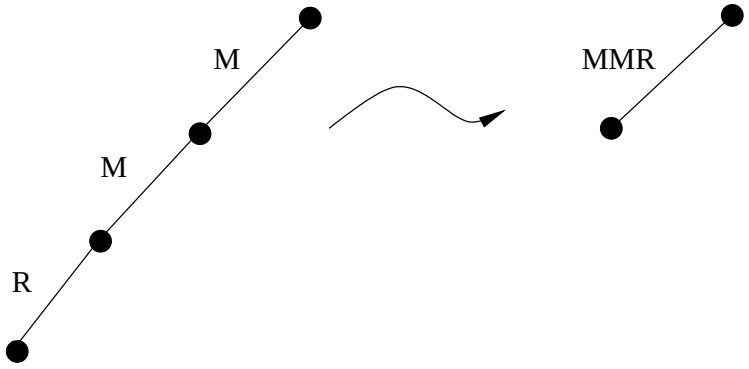
Eigenschaften des Tries:

- ▷ Das Substringproblem $q \overset{?}{<}_s w$ kann in $O(|q|)$ Schritten gelöst werden.

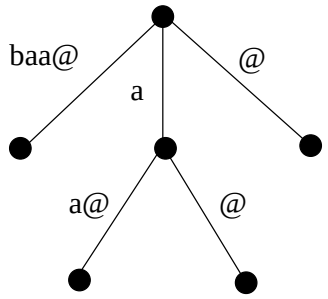
- ▷ Konstruktion des Trie(w) ist naiv in $O(|w|^2)$ Schritten möglich. Mit Algorithmen nach McCreight oder Weiner geht es auch in $O(|\text{Trie}(w)|)$ Schritten ($O(|\text{Trie}(w)|) \sim$ Anzahl Knoten).

1.5.1.2 Suffix–Tries

Übergang zur kompakteren Darstellung durch Zusammenfassung von Pfaden im Baum, entlang denen jeder Knoten den Ausgrad 1 hat:



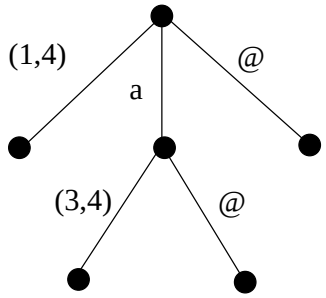
Es entsteht SuffixTrie(w) — im obigen Beispiel:



Eigenschaften des Suffix-Tries:

- ▷ Da jeder innere Knoten Ausgrad ≥ 2 hat (Ausnahme: $w = @$) und es $|w|$ Blätter gibt, hat SuffixTree(w) $O(|w|)$ Knoten.
- ▷ Durch Erweiterung von McCreights (oder Weiners) Algorithmus ist Konstruktion in $O(|w|)$ Schritten möglich.
- ▷ Im worst-case (z.B. $w = a^n b^n @$) kostet die Kantenbeschriftung $O(|w|^2)$ Speicher. Einfacher Ausweg: Speichere anstatt jeder Kantenbeschriftung der Länge ≥ 2 die entsprechende Start- & End-Position im Wort w.

  1.22 Beispiel $w = w_1 w_2 w_3 w_4 = baa@$



  → Speicherbedarf $O(|w| \log |w|)$ Bits.

- An den Blättern speichert man i.a. die Startposition des entsprechenden Suffixes von w.

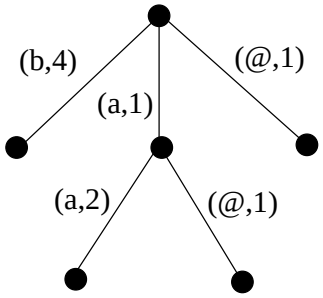
1.5.1.3 PATRICIA-Tries (Practical Algorithm to Retrieve Information Coded in Alphanumeric)

Ziel: Vermeidung von Plattenzugriffen während des Suchens bei jedem internen Knoten (Annahme: w passt nicht ganz in den Hauptspeicher).
Idee: Verwende Baumstruktur von SuffixTrie(w) und speichere zu jeder Kante e anstelle der Beschriftung $e_1 \dots e_m$

1. das erste Zeichen e_1 der Kantenbeschriftung und

2. die Länge m der Kantenbeschriftung.

Nun Kantenbeschriftungen aus $(\Sigma \cup \{@\}) \times \mathbb{N}$.
Beispiel: PatriciaTrie(w):



Suche in PatriciaTrie(w)**:** $q \overset{?}{<}_s w$. (Grobversion)

1. Durchlaufe PatriciaTrie(w) von der Wurzel an und benutze Pointer $1 \leq p \leq |q|$ auf die aktuelle Position im Query q (Start: $p \leftarrow 1$).

2. Ist $p \leq |q|$ und gibt es Kante $v \rightarrow v'$ mit Beschriftung (x, k) ausgehend vom momentanen Knoten v wobei $q_p = x$,

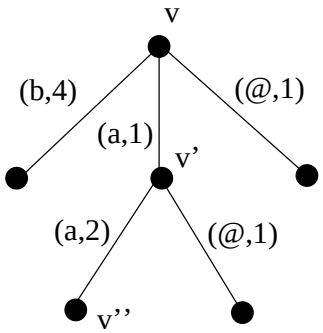
   - so verzweige zum Knoten v' und setze $p \leftarrow p + k$.
   - Falls keine solche Kante existiert, ist $q \not<_s w$.

3. Ist $p > |q|$, überprüfe durch Zugriff auf w, ob an einer der lokalisierten Positionen tatsächlich q in w vorkommt.

   Hierzu müssen die zu überprüfenden Positionen in w aus den (evtl. allen) Blättern, die Nachfolger des aktuellen Knotens sind, abgelesen werden.

   (Notwendig, da ja i.a. Zeichen übersprungen wurden.)

1.23 Beispiel Suche $q = q_1q_2q_3 = aaa$ in $w = baa@$



Hier verfolgt man den Weg $v \rightarrow v' \rightarrow v''$ und matcht die Zeichen q_1 und q_2 von q. Im Blatt v'' testet man, ob $w[2:4] \overset{?}{=} q$ und bemerkt, dass dies *nicht* der Fall ist, also $q \not<_s w$.

1.5.1.4 Directed Acyclic Word Graphs (DAWGs)

Beim Übergang Trie → Suffix-Trie hatten wir *Kantenbeschriftungen* zusammengefaßt.

Idee hier: Fasse *isomorphe Subgraphen* des Tries zusammen.

Für einen Substring $x <_s w$ sei durch

$$e_w(x) := \{i \mid x = w[i - |x| + 1 : i]\}$$

die Menge der Endpositionen aller Vorkommen von x in w bezeichnet. Zwei Wörter $x, y \in \Sigma^*$ heißen *rechtsäquivalent*, $x \equiv_w y$, gdw. $e_w(x) = e_w(y)$. Die Äquivalenzklassen bezeichnen wir hier mit

$$[x] := \{y \in \Sigma^* \mid x \equiv_w y\}.$$

1.24 Beispiel $w = abcbc$. Dann ist $e_w(bc) = \{3, 5\} = e_w(c)$ und $[bc] = \{bc, c\}$.

1.25 Bemerkung

- In jeder Äquivalenzklasse $[x]$ gibt es ein eindeutig bestimmtes Element $\overline{x} \in \Sigma^*$ maximaler Länge.
- $x \equiv_w y \Rightarrow x$ ist Suffix von y oder y ist Suffix von x. (Es gilt i.a. *nicht* $\Leftarrow$.)

1.26 Definition Sei $w \in \Sigma^*$. DAWG$(w) = (V, E, \alpha)$ ist der Graph mit

1. Knotenmenge $V := \{\overline{x} \mid x <_s w\} \subseteq \Sigma^*$,
2. Kantenmenge $E := \{(x, y) \in V \times V \mid \exists a \in \Sigma \sqcup \{@\} : [xa] = [y]\}$ und
3. Kantenbeschriftung $\alpha : E \to \Sigma \sqcup \{@\}$, $E \ni (\overline{x}, \overline{xa}) \mapsto a$.

◇

Bemerkung: $\{\varepsilon\} = [\varepsilon]$ ist Äquivalenzklasse mit maximalem Element ε. Der Knoten ε im DAWG hat stets Eingrad 0.

1.27 Beispiel Für $w = baa@$ erhält man als Menge der Substrings

$$\{\varepsilon, a, b, @, ba, aa, a@, baa, aa@, baa@\}$$

und als Äquivalenzklassen

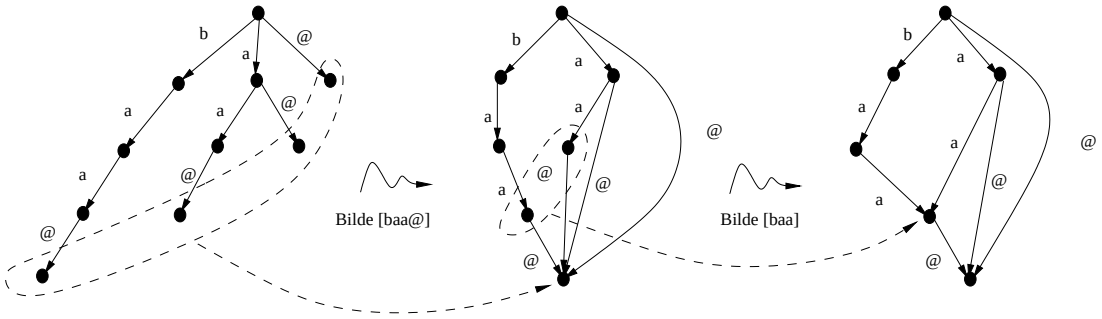
$[\varepsilon] = \{\varepsilon\}, [a] = \{a\}, [b] = \{b\}$

$[ba] = \{ba\}$

$[baa] = \{aa, baa\}$

$[baa@] = \{@, a@, aa@, baa@\}$

Umformung unseres Beispiel-Tries in einen DAWG:



Vorteile von DAWGs:

- Nur ein Zeichen pro Kante (einfacher im Rechner zu verwalten als Zeichenkette variabler Länge),
- Größe ist beschränkt durch
  - $|V| < 2|w|$ und
  - $|E| < |V| + |w| - 1$ (nichttriviales Resultat).

Nachteil:

- Relativ kompliziert zu konstruieren.

1.5.1.5 Suffix–Arrays

Zum Abschluß betrachten wir noch eine relativ einfache Indexvariante, die zwar eine etwas längere Suchzeit liefert, sich aber in der Praxis aufgrund ihrer Einfachheit und Effizienz in realen Anwendungen bewährt hat.

Ideen:

- Jeder Substring $q <_s w$ ist Präfix eines Suffixes von w.
- Sortiere alle Suffixe von w lexikographisch und suche q dann binär.

Grobvorgehen:

- $w =: w_1 \dots w_n$. Für $1 \leq i \leq n$ sei $w^i = w_i \dots w_n$. Sei $s_1, \dots, s_n$ Permutation von $[1 : n]$, so dass

$$w^{s_1} <_{\text{lex}} w^{s_2} <_{\text{lex}} \dots <_{\text{lex}} w^{s_n},$$

  wo $<_{\text{lex}}$ die lexikographische Ordnung auf Σ^+,

$$\begin{aligned} v <_{\text{lex}} w \quad :\Leftrightarrow \quad & (v = w[1 : |v|] \ \wedge \ v \neq w) \\ & \vee \ (\exists 1 \leq i \leq |v| : v[1 : i-1] = w[1 : i-1] \ \wedge \ v_i <_\Sigma w_i) \end{aligned}$$

  für eine feste totale Ordnung $<_\Sigma$ auf Σ ist.

- Binärsuche von q in $(w^{s_1}, \ldots, w^{s_n})$:
  - Gilt $w^{s_i} <_{\text{lex}} q$, ist q nicht Präfix von w^{s_i} und die Suche geht „oberhalb“ von w^{s_i} binär weiter.
  - Gilt $q <_{\text{lex}} w^{s_i}$, teste ob q Präfix von w^{s_i} ist.

    Falls ja: Ausgabe $q <_s w$.

    Falls nein: Suche „unterhalb“ von w^{s_i} binär weiter.
  - Terminierung, falls kein weiterer binärer Suchschritt möglich ist (dann ist $q \not<_s w$).

1.28 Bemerkung

▷ $q <_s w$ ist mit Suffix-Array so (naiv) in $O(|q| \log |w|)$ Schritten entscheidbar.

▷ Verfeinerter Algorithmus (weitere Vorverarbeitung von w basierend auf einer Berechnung längster gemeinsamer Präfixe der $(w^i)_i$) liefert eine Laufzeit von $O(|w| + |q|)$.

Nachbemerkung zu diesem Abschnitt: Problematisch ist bei den vorgestellten Verfahren zum stringbasierten Retrieval meist der Bereich Fehlertoleranz.
→ Wie löst man etwa das Substringproblem mit k-Mismatches (siehe 1.2)?

1.5.2 Termbasiertes Retrieval

1.5.2.1 Invertierte Listen

Grundprinzip ist eine „Inversion“ der normalen Datenrepräsentation eines Texts (als Folge):

	1	2	3	4	5
Textausschnitt:	der	Tiger	auf	der	Jagd

Wir haben es hier mit einer Abbildung $\sigma : [1:5] \to T \subset L^+$ zu tun,

$$1 \mapsto \text{ der}, 2 \mapsto \text{ Tiger}, \ldots, 5 \mapsto \text{ Jagd}.$$

σ ist natürlich i.a. nicht injektiv!
Invertierter Index dieses Textausschnitts:

der	$\mapsto$	1,4
Tiger	$\mapsto$	2
auf	$\mapsto$	3
Jagd	$\mapsto$	5

Hier haben wir eine Abbildung $T \to 2^{[1:5]}$, $t \mapsto \sigma^{-1}(t)$, also eine Inversion von σ. Dem Term t werden hier alle Positionen seines Auftretens im Text zugewiesen. Dieses Prinzip ist bereits aus Abschnitt 1.3 bekannt (Mengen $L_{\mathcal{D}}$).

Betrachten hier invertierte Listen in einem für das Dokumentenretrieval wichtigen **Spezialfall**. (Viel mehr zu Varianten von invertierten Listen in den Kapiteln 3 und 4.)

Sei $\mathcal{D} = (D_1, \ldots, D_N)$ Dokumentenkollektion wo $\forall i : D_i \subseteq L$ und $\tau : L \to T$ Termextraktor mit endlicher Termmenge $T \subseteq L$.

1.29 Definition Für $t \in T$ heißt die Menge

$$L_{\mathcal{D}}(t) := \{i \in [1:N] \mid t \in \tau(D_i)\}$$

invertierte Liste zu t (bezüglich $\mathcal{D}$). ◇

Boolesche Anfragen lassen sich analog zum Vorgehen aus 1.2 mit Hilfe der invertierten Listen beantworten:

1.30 Lemma Sei $q \in Q(T)$ boolesche Anfrage, dann gilt die Rekursion

$$B_{\mathcal{D}}(q) = \begin{cases} L_{\mathcal{D}}(q) & \text{falls } q \in T, \\ B_{\mathcal{D}}(p) \cup B_{\mathcal{D}}(r) & \text{falls } q = (p \vee r), \\ B_{\mathcal{D}}(p) \cap B_{\mathcal{D}}(r) & \text{falls } q = (p \wedge r), \\ [1:N] \backslash B_{\mathcal{D}}(p) & \text{falls } q = (\neg p). \end{cases}$$

◇

Beweis: Übung!

1.31 Beispiel Betrachten extrahierte Termmengen

$\tau(D_1) = \{$ Retrieval, Kurth, Clausen $\}$ und $\tau(D_2) = \{$ Audio, Retrieval, Kurth $\}$

zu Kollektion aus Dokumenten D_1 und D_2, sowie die Anfrage

$$q := ((\text{Kurth} \wedge \text{Retrieval}) \wedge (\neg \text{Clausen})).$$

Die Trefferauswertung liefert wegen

$$B_{\mathcal{D}}(\text{Kurth}) = \{1,2\}, B_{\mathcal{D}}(\text{Retrieval}) = \{1,2\}, B_{\mathcal{D}}(\text{Clausen}) = \{1\}$$

und

$$B_{\mathcal{D}}((\text{Kurth} \wedge \text{Retrieval})) = \{1,2\} \text{ sowie } B_{\mathcal{D}}((\neg \text{Clausen})) = \{2\}$$

schließlich $B_{\mathcal{D}}(q) = \{2\}$.

Anfragen im Vektorraumretieval:

Hier verwendet man zur Termfolge $T =: (t_1, \ldots, t_M)$ die um Termgewichte w_{ij} erweiterten invertierten Listen

$$L^V_{\mathcal{D}}(t_i) := \{(j, w_{ij}) \mid t_i \in \tau(D_j)\}.$$

w_{ij} ist Gewicht zu Term t_i in Dokument j. (Ist $w_{ij} = 0$, speichert man keinen Eintrag (j, w_{ij})).

Die Anfrageauswertung hängt nun von der Retrievalfunktion ab!
Betrachten hier das innere Produkt, $\mathrm{rsv}(q, D_j) := \langle (w_{iq}) | (w_{ij}) \rangle$ für einen Anfragevektor $(w_{1q}, \ldots, w_{Mq}) \in \mathbb{R}^M$ zur Anfrage q.
Definiert man für $w \in \mathbb{R}$ und $t_i \in T$

$$w \cdot L^V_{\mathcal{D}}(t_i) := \{(j, w \cdot w_{ij}) \mid (j, w_{ij}) \in L^V_{\mathcal{D}}(t_i)\}$$

sowie

$$\begin{aligned} L^V_{\mathcal{D}}(t) \overset{V}{\cup} L^V_{\mathcal{D}}(s) &:= \{(j, v+w) \mid (j,v) \in L^V_{\mathcal{D}}(t) \ \wedge \ (j,w) \in L^V_{\mathcal{D}}(s)\} \\ &\cup \{(j,v) \mid (j,v) \in L^V_{\mathcal{D}}(t) \wedge \ \nexists w : (j,w) \in L^V_{\mathcal{D}}(s)\} \\ &\cup \{(j,w) \mid \ \nexists v : (j,v) \in L^V_{\mathcal{D}}(t) \ \wedge \ (j,w) \in L^V_{\mathcal{D}}(s)\} \end{aligned}$$

(sowie entsprechendes für „mit Gewichten w_{ij} multiplizierte“ Listen), so liefert

$$\bigcup_{i \in [1:N], w_{iq} \neq 0}^{V} w_{iq} \cdot L^V_{\mathcal{D}}(t_i) \quad \hat{=} \quad \sum_{i=1}^{N} w_{iq} \cdot \Lambda_i$$

die rsv-Werte zur Anfrage q bzgl. des inneren Produktes (Übung!).

Bemerkung: $L^V_{\mathcal{D}}(t_i)$ ist eigentlich der Graph der Funktion

$$\begin{aligned} \Lambda_i : [1:N] &\to \mathbb{R} \\ j &\mapsto w_{ij}. \end{aligned}$$

$L^V_{\mathcal{D}}(t_i) \overset{V}{\cup} L^V_{\mathcal{D}}(t_k)$ ist gerade der Graph der Funktion

$$\Lambda_i + \Lambda_k := (j \mapsto w_{ij} + w_{kj}).$$

◇

Effiziente Bestimmung der Treffermenge

Konzentrieren uns hier auf das boolesche Retrieval.

1. Man speichert die $L_{\mathcal{D}}(t)$ als — o.B.d.A. aufsteigend — sortierte *Folgen* ab (daher ist der gebräuchliche Name auch invertierte *Liste* und nicht invertierte Menge).

   Die Sortierung erfolgt bei der Indexierung bzw. beim Datenupdate.

2. Die Auswertung boolescher Anfragen erfordert dann folgende Operationen auf sortierten Folgen:

   - Ist $q = (p \vee r)$, so muss $B_{\mathcal{D}}(p) \cup B_{\mathcal{D}}(r)$ gebildet werden. Also müssen die $B_{\mathcal{D}}(p)$ und $B_{\mathcal{D}}(r)$ entsprechenden Folgen verschmolzen werden. Seien $n := |B_{\mathcal{D}}(p)|$ und $m := |B_{\mathcal{D}}(r)|$.

  - Ist $n \approx m$, empfiehlt sich ein lineares Mergen mit Kosten $O(n+m)$.
  - Ist $n \ll m$, empfiehlt sich ein Mergen durch binäres Einfügen der Elemente von $B_{\mathcal{D}}(p)$ (kurze Liste) in $B_{\mathcal{D}}(r)$ (lange Liste) mit Kosten $O(n \log m)$.
- Ist $q = (p \wedge r)$, so muss $B_{\mathcal{D}}(p) \cap B_{\mathcal{D}}(r)$ gebildet werden. Also muss die dem Schnitt von $B_{\mathcal{D}}(p)$ und $B_{\mathcal{D}}(r)$ entsprechende Folge gebildet werden. Dies geschieht wieder mit Varianten obiger Algorithmen:
  - Ist $n \approx m$, durchlaufe $B_{\mathcal{D}}(p)$ und $B_{\mathcal{D}}(r)$ wie beim Mergen, füge jedoch nur die in beiden Listen vorkommenden Elemente in den Schnitt ein. Kosten: $O(n + m)$.
  - Ist $n \ll m$, binäre Suche der n Elemente der kleineren Liste in der größeren. Nur gefundene Elemente werden in die Resultatfolge eingefügt. Kosten: $O(n \log m)$.
- Komplementbildung geht in $O(N)$ Schritten (linear in der Anzahl aller Dokumente).

Wichtige **Invariante**: Vor jedem Schritt ist das Zwischenergebnis wieder eine sortierte Folge.

Genaueres zur Laufzeit folgt später für Varianten invertierter Listen im Zusammenhang mit speziellen Anfragetypen.
Nun behandeln wir noch einige für die Praxis relevante Fragestellungen:

- Indexkompression,
- Verwaltung der Listen, Zugriff auf das „Lexikon“,
- Indexaufbau.

Indexkompression

Wir schreiben die invertierten Listen als aufsteigende Folgen

$$L_{\mathcal{D}}(t) =: (e_1, e_2, \ldots, e_n),$$

$e_1 < e_2 < \ldots < e_n$.
Zunächst Idee: Codiere Differenzenfolge

$$(d_1, \ldots, d_n) := (e_1, e_2 - e_1, e_3 - e_2, \ldots, e_n - e_{n-1})$$

mit Werten $d_i \in [1 : N]$. Man erhofft, dass die d_i i.a. klein sind.
Eine „optimale“ Codierung bzgl. eines minimalen Bitbedarfs zur Speicherung der codierten Symbole erhält man durch Betrachtung der Häufigkeiten der zu codierenden Symbole $[1 : N]$. Wir betrachten hier nur binäre Codierungen der Symbole des Alphabets $\Omega = [1 : N]$.

1.32 Definition Sei Ω eine endliche Menge von Symbolen.

1. Eine Abbildung $C : \Omega \to \{0,1\}^+$ heißt *binärer Code* zu Ω.
2. C heißt *eindeutig decodierbar*, falls seine Fortsetzung $\Omega^+ \to \{0,1\}^+$ injektiv ist.
3. C heißt *Präfixcode*, falls kein Codewort Präfix eines anderen ist:

$$\forall x, y \in \Omega : C(x) = ab \wedge C(y) = a \Rightarrow b = \varepsilon \wedge x = y.$$

1.33 Bemerkung

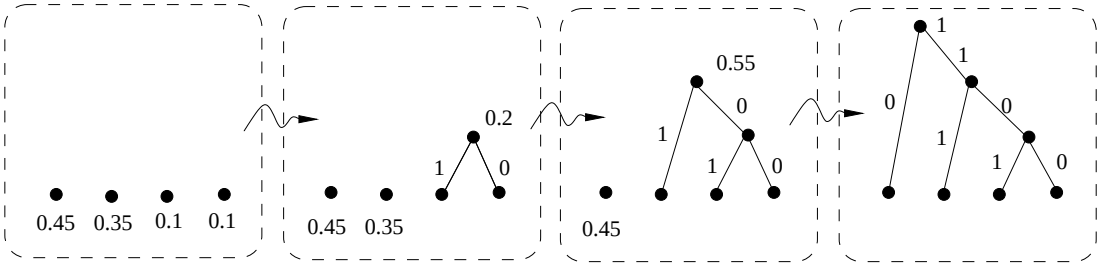
Es gibt eindeutig decodierbare Codes, die keine Präfixcodes sind. (Übung!)

Wir interessieren uns hier für Präfixcodes, da wir somit Folgen von Codewörtern „von links nach rechts“ decodieren können ohne vorab die ganze Folge lesen zu müssen. ◇

1.34 Beispiel Huffman-Codes sind Präfixcodes. Sei für $\Omega := [1 : n]$ durch $p_1, \ldots, p_n$ eine Folge von W-Gewichten (mit $\sum_{i=1}^n p_i = 1$) der Symbole aus Ω gegeben, dann konstruiert man den zugehörigen binären Huffman-Code wie folgt mittels eines binären Baums (V, E) mit Knotenbeschriftung $p : V \to [0,1]$ und Kantenbeschriftung $C : E \to \{0,1\}$.

1. $v_1, \ldots, v_n$ seien Knoten mit Beschriftung $p(v_i) := p_i$. Setze $V := \{v_1, \ldots, v_n\}$ und $E := \emptyset$.
2. V' sei die Menge der aktuell zu betrachtenden Knoten. Wir starten mit $V' = V$.
3. Solange $|V'| > 1$
   (a) $v' := \text{argmin}_{v \in V'} p(v)$,
   (b) $v'' := \text{argmin}_{v \in V' \setminus \{v'\}} p(v)$,
   (c) $V' := V' \setminus \{v', v''\}$,
   (d) $V' := V' \cup \{v^*\}$ für einen neuen Knoten v^*,
   (e) $V := V \cup \{v^*\}$,
   (f) $p(v^*) := p(v') + p(v'')$,
   (g) $E := E \cup \{(v^*, v'), (v^*, v'')\}$.
   (h) $C(v^*, v') := 0$, $C(v^*, v'') := 1$.
4. Die Fortsetzung von C auf alle Pfade ℓ_i von der Wurzel des so konstruierten Baums zu einem Blatt v_i liefert einen Huffman-Code $\Omega \ni i \mapsto C(\ell_i)$.

Zahlenbeispiel: Seien $p_1 := 0.45, p_2 := 0.35, p_3 := 0.1, p_4 := 0.1$ W-Gewichte, dann kann man obige Baumkonstruktion wie folgt visualisieren:



In der Grafik sind hier für jeden Schritt nur die Gewichte $p(v)$ der Knoten aus V' dargestellt. Sind in einem Konstruktionsschritt die Knoten mit minimalen Gewichten nicht eindeutig bestimmt, wählt man unter den Knoten minimalen Gewichts zwei beliebige aus. Im obigen Beispiel ergibt sich $C(v_1) = 0, C(v_2) = 11, C(v_3) = 101, C(v_4) = 100$. ◇

1.35 Bemerkung

Huffman-Codes sind (asymptotisch) sogar bzgl. der Anzahl benötigter Codierungs-Bits optimal.

Nachteile von Huffman-Codes (bzgl. der Codierung invertierter Listen): Speicherung des Huffman-Baums nötig, W-Statistik muss erstellt werden (und diese kann sich beim Dokumentenupdate ändern), N ist i.a. vorab nicht beschränkt. ◇

In unserem Anwendungsfall kann man auch ein einfacheres Modell verwenden, das zudem mit unbeschränktem N funktioniert. Betrachten hierzu die Wahrscheinlichkeit für das Auftreten eines Intervallsprungs d_i in einer beliebigen Liste:
Ist

$$f := \sum_{t \in T} |L_{\mathcal{D}}(t)|$$

die Anzahl aller Einträge in den invertierten Listen ($f \leq N \cdot |T|$), dann ist

$$p := \frac{f}{|T| \cdot N} \in [0, 1]$$

die Wahrscheinlichkeit, dass ein beliebiger Term in einem beliebigen Dokument vorkommt (bei Gleichverteilung der Terme auf die Dokumente). Betrachten wir die Listeneinträge, dann ist mit dieser Überlegung

$$P(k) := (1 - p)^{k-1} p$$

die Wahrscheinlichkeit, dass ein Term in $k-1$ aufeinanderfolgenden Dokumenten nicht vorkommt, dafür aber im k-ten Dokument (k Bernoulli-Experimente). Dies liefert eine geometrische W-Verteilung (Übung!)

$$k \mapsto P(k)$$

auf den Abständen d_i der Dokumentennummern.
Ein Code, der diese Verteilung gut approximiert ist der sogenannte *Golomb Code* (nach Solomon Golomb, 1966). Dazu sei $b \in \mathbb{N}$ ein geeigneter Codeparameter. Ist $k \in \mathbb{N}$ zu codieren, bildet man zunächst $q := \lfloor \frac{k-1}{b} \rfloor$ und codiert dann via

$$C_b : k \mapsto (\text{unär}(q+1), C(k - qb - 1)),$$

wobei C eine geeignete präfixfreie Codierung des Residualwertes $k - qb - 1$ mit Codewörtern der Länge $\lceil \log b \rceil$ und $\lfloor \log b \rfloor$ Bits ist.

Für $k \in \mathbb{N}$ ist unär(k) die Unärcodierung

$$\mathbb{N} \ni k \mapsto 1^{k-1}0.$$

Beispiel: Sei $b = 3$, dann gibt es die Residualwerte $0, 1$ und 2. Diese codiert man mit dem Code

$$C : 0 \mapsto 0, 1 \mapsto 10, 2 \mapsto 11.$$

Ist $b = 6$, erhält man mögliche Residualwerte aus $[0 : 5]$, die man mit

$$C : 0 \mapsto 00, 1 \mapsto 01, 2 \mapsto 100, 3 \mapsto 101, 4 \mapsto 110, 5 \mapsto 111$$

codiert.
Golomb-Codes für $b = 3$ und $b = 6$:

k	Unärcodierung	Golomb-Codierung	
		$b = 3$	$b = 6$
1	0	0 0	0 00
2	10	0 10	0 01
3	110	0 11	0 100
4	1110	10 0	0 101
5	11110	10 10	0 110
6	111110	10 11	0 111
7	1111110	110 0	10 00
8	11111110	110 10	10 01
9	111111110	110 11	10 100
10	1111111110	1110 0	10 101
11	11111111110	1110 10	10 110
12	111111111110	1110 11	10 111

1.36 Bemerkung

- Die Codierung der Residualwerte ist so zu treffen, dass die Codes präfixfrei sind. In den obigen Beispielen ist ein allgemeines Konstruktionsprinzip für die Codierung zu erkennen, wobei man für die b Residualwerte einen Huffmancode verwendet. Man erhält für $b = 3$ und $b = 6$ die in Abb. 1.5 dargestellten Huffmancodes.

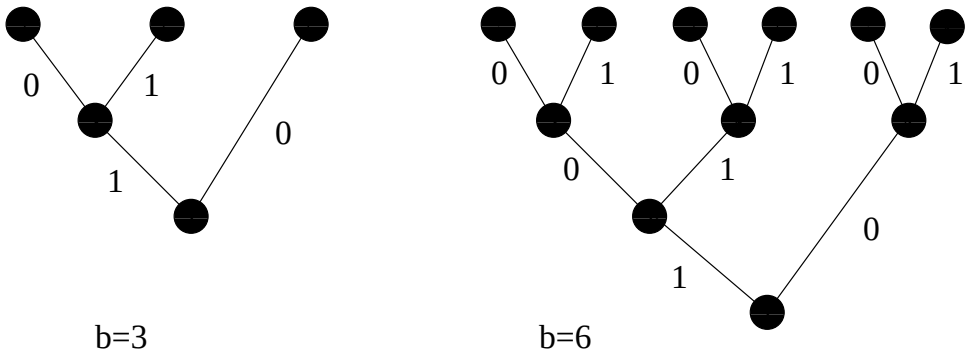


Abbildung 1.5: Huffmancodes für die Residualwerte bei der Golomb-Codierung für $b = 3$ (links) und $b = 6$ (rechts).

- Wählt man b so, dass

$$(1-p)^b + (1-p)^{b+1} \leq 1 < (1-p)^{b-1} + (1-p)^b,$$

dann liefert C_b einen bezüglich der Anzahl benötigter Bits optimalen binären Code, der zudem präfixfrei ist. Codes dieser Art nennt man *run-length codes*. (Übung: Warum ist die Wahl eines solchen b's immer möglich?)

- Ein weiteres bekanntes Beispiel ist der γ-*Code*

$$\mathbb{N} \ni k \mapsto (\text{unär}(1 + \lfloor \log k \rfloor), \text{binär}(k - 2^{\lfloor \log k \rfloor})),$$

wobei die Residualwerte mit $\lfloor \log k \rfloor$ Bits binär (mit führenden Nullen) codiert werden.

k	Unärcodierung	γ-Codierung
1	0	0
2	10	10 0
3	110	10 1
4	1110	110 00
5	11110	110 01
6	111110	110 10
7	1111110	110 11
8	11111110	1110 000

Das Top-Down Konstruktionsprinzip (siehe Tabelle) für die Residualwerte ist hier, für den unären Präfix 1^n0 jeweils alle binären n-Tupel (mit führenden Nullen) in lexikographischer Sortierung zu verwenden. Somit gibt es 2^n Codewörter der Länge $(n+1) + n = 2n + 1$.

- Allgemeines Prinzip dieser Art von Codes: Wähle Folge $(v_i)_{i\in\mathbb{N}}$ von Intervallen. Zu codierendem $k \in \mathbb{N}$ sucht man eindeutig bestimmtes $\ell \in \mathbb{N}$, s.d.

$$\sum_{i=1}^{\ell} v_i < k \leq \sum_{i=1}^{\ell+1} v_i$$

und codiert

$$k \mapsto (\text{unär}(\ell), C(k-1-\sum_{i=1}^{\ell} v_i)),$$

wobei der Residualwert $k-1-\sum_{i=1}^{\ell} v_i$ geeignet mit $\lceil \log v_{\ell+1} \rceil$ Bits codiert werden kann.

Beispiel: Für den γ-Code hat man $v_i = 2^{i-1}$ für $i \geq 1$ und Residualcodierung $C(k-1-\sum_{i=1}^{\ell} v_i) = \text{binär}(k - 2^{\lfloor \log k \rfloor})$.

$\diamond$

Verwaltung der Listen, Zugriff auf das „Lexikon“

Listenverwaltung ist wichtig, da es u.U. sehr viele verschiedene Terme ≡ Listen geben kann.
Skizzieren hier nur ein paar Möglichkeiten, die Informationen zu jeder Liste — i.a.

$$(t, \text{tf}_t, p_t) \in T \times \mathbb{N} \times \mathbb{N}$$

bestehend aus Term t, Summe $\text{tf}_t := \sum_{i=1}^{N} \text{tf}(t, D_i)$ der Vorkommen von Term t und Adresse p_t der Liste zu Term t — zu verwalten.
Diese Informationen (plus zugehörige Speicherstrukturen) werden auch als *Lexikon* bezeichnet.
Möglichkeiten zur platzsparenden Organisation des Lexikons:

1. **Speichere Folge von Records**

$$(t, \text{tf}_t, p_t)_{t\in T}$$

Beispiel:

$$\ldots (\text{Frank}, 100, 343), (\text{Franz}, 45, 123), (\text{Franzose}, 15, 32), \ldots$$

Nachteil: Speicher-Overhead bei der Verwaltung der Strings.

2. **Konkatenation aller Terme zu einem String**

Sei $T = (t_1, \ldots, t_M)$.

(a) Bilde $\sigma := t_1 t_2 \dots t_M \in \Sigma^+$ und für $i \in [1:M]$

$$a_i := 1 + \sum_{j=1}^{i-1} |t_j|,$$

dann ist $\sigma[a_i : a_{i+1} - 1] = t_i$.

(b) Speichere σ und $(a_i, \mathrm{tf}_{t_i}, p_{t_i})_{i \in [1:M]}$.

1.37 Beispiel

$$\begin{matrix} & & 570 & & 575 & & 580 & \\ (\dots & \mathrm{Frank} & \mathrm{Franz} & \mathrm{Franzose} & \dots) = \sigma & & & \end{matrix}$$

$$\dots (570, 100, 343), (575, 45, 123), (580, 15, 32) \dots$$

Nachteil: Speicheraufwand für Pointer a_i ist eventuell groß.

3. **Speichere nur Pointer auf jeden L-ten Term von σ**

1.38 Beispiel $L := 2$

$$\begin{matrix} & & \overset{A_{2i}}{\downarrow} & & & & \overset{A_{2(i+1)}}{\downarrow} & & \\ (\dots & 5 & \mathrm{Frank} & 5 & \mathrm{Franz} & 8 & \mathrm{Franzose} & \dots) =: \sigma' \end{matrix}$$

$A_i \equiv$ Adresse des i-ten Terms in σ'.

In einer anderen Liste speichert man die $(\mathrm{tf}_{t_i}, p_{t_i})_{i \in [1:M]}$-Paare.

4. **Front-Coding**

Idee: Zu aufeinanderfolgenden Termen t_i, t_{i+1} speichere Länge ℓ_{i+1} des längsten gemeinsamen Präfixes $\mathrm{lcp}(t_i, t_{i+1}) \in \Sigma^*$.

$|t_{i+1}| - \ell_{i+1}$ ist dann Restlänge von t_{i+1}, die mit t_{i+1} zusammen explizit gespeichert werden muss.

Speichere dann

$$(\ell_i, |t_i| - \ell_i, t_i[\ell_i + 1 : |t_i|])_{i \in [2:M]},$$

sowie t_1 explizit (plus $(\mathrm{tf}_{t_i}, p_{t_i})_{i \in [1:M]}$ wie oben).

Beispiel: Statt eines Terms t_{i+1} =Franz speichert man $(4, 1, \mathrm{z})$, falls der vorherige Term t_i =Frank war.

5. **Minimal Perfect Hashing**

Idee: Kennt man eine effizient berechenbare und *kompakt darstellbare* Bijektion

$$h : T \to [0 : |T| - 1],$$

dann müssen die Terme bei festem T *nicht* mehr explizit gespeichert werden.

Klar ist, dass solch ein h existiert — oben haben wir bereits gesehen, wie wir durch explizites Abspeichern der Terme als Strings h als Tabelle $(t, h(t))$ darstellen könnten. Wie kommt man jedoch *ohne* Speicherung der t's aus?

h kann als **Hashfunktion** angesehen werden. Ist h injektiv, so heißt h *perfekt*. Ist h sogar eine Bijektion, heißt h *minimal perfekt*.

Grobskizze eines randomisierten Algorithmus zum Auffinden eines solchen h:

(a) Sei $\pi : \Sigma \to [0 : |\Sigma| - 1]$ eine Bijektion, die jedem Zeichen $c \in \Sigma$ einen Zahlenwert $\pi(c)$ zuordnet. (z.B. ASCII-Codierung)

(b) Sei $\ell :=\max_{t \in T} |t|$ die maximale Termlänge in T.

(c) Solange keine geeignete Funktion h gefunden wurde:

i. Wähle geeigneten kleinen Parameter $m \geq |T|$,

ii. wähle zufällig zwei Gewichtsvektoren $w_1, w_2 \in [0 : m-1]^\ell$ und

iii. definiere hiermit zwei Hashfunktionen: Für $j \in [1 : 2]$ sei

$$\begin{aligned} h_j : T &\to [0 : m-1] \\ t &\mapsto \left(\sum_{i=1}^{|t|} \pi(t[i]) \cdot w_j[i] \right) \bmod m. \end{aligned}$$

iv. Versuche eine Funktion

$$g : [0 : m-1] \to [0 : |T| - 1]$$

zu finden, so dass

$$\begin{aligned} h' : T &\to [0 : |T| - 1] \\ t &\mapsto (g(h_1(t)) + g(h_2(t))) \bmod |T| \end{aligned}$$

eine Bijektion ist. Gelingt dies, setze $h := h'$ und der Algorithmus terminiert erfolgreich.

1.39 Beispiel (aus *Managing Gigabytes*) $|T| = 12, \ell = 11, m = 15$

Term t	$h_1(t)$	$h_2(t)$	$h(t)$
jezebel	5	9	0
jezer	5	7	1
jezerit	10	12	2
jeziah	6	10	3
jeziel	13	7	4
jezliah	13	11	5
jezoar	4	2	6
jezrahiah	0	3	7
jezreel	6	3	8
jezreelites	8	4	9
jibsam	9	14	10
jidlaph	3	1	11

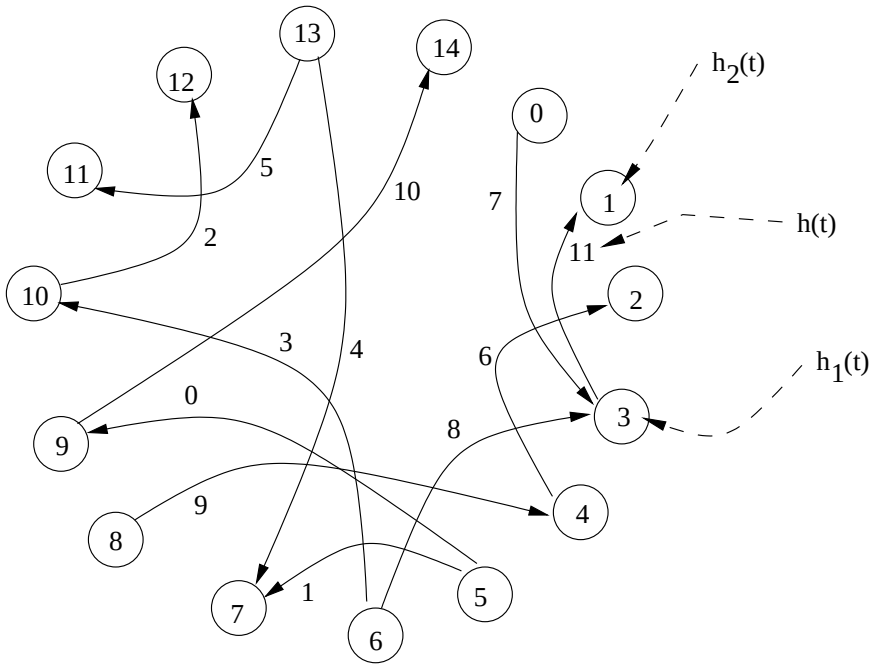
x	$g(x)$
0	0
1	4
2	0
3	7
4	6
5	0
6	1
7	1
8	3
9	0
10	2
11	2
12	0
13	3
14	10

Konstruktion von h ist graphentheoretisch möglich:

Bilde gerichteten Graphen G aus Knoten $V := [0 : m-1]$ und

Kanten $E := \{(h_1(t), h_2(t)) \mid t \in T\}$.

Falls $|E| = |T|$, wähle als Kantengewichtung Bijektion $\alpha : E \to [0 : |T|-1]$. (Ansonsten wähle solange neue w_1, w_2 ($\to$ neue Hashfunktion) bis $|E| = |T|$ erfüllt ist.)



Übung: Ist der durch G induzierte ungerichtete Graph azyklisch, kann man g iterativ anhand der Zusammenhangskomponenten von G bestimmen!

(Das geht so: Betrachte Zusammenhangskomponente von G. Beginne mit Quellknoten x vom Eingrad 0. Bilde x ab auf beliebigen g-Wert. Damit sind alle g-Werte für diese Zusammenhangskomponente eindeutig festgelegt: $x \xrightarrow{\xi} y \Rightarrow g(y) = \xi - g(x) \bmod |T|$ und $z \xrightarrow{\zeta} x \Rightarrow g(z) = \zeta - g(x) \bmod |T|$.)

1.40 Bemerkung

- ▷ Terminierung des Algorithmus umso wahrscheinlicher, je größer m ist.
- ▷ Man kann zeigen, dass jedes in Frage kommende h mindestens $1.44|T|$ Bits Speicherplatz (kompakte Beschreibung der Funktion) benötigt.
- ▷ Für genaueres zu erwarteten Laufzeiten des Algorithmus, siehe *Managing Gigabytes*, Kapitel 4.1.

Zum Abschluss noch ein Vergleich des Speicherplatzbedarfs der gerade beschriebenen Methoden zur Lexikonverwaltung: (hier ist $|T| = 1.000.000$)

Methode	**Speicherplatz [MB]**
1. Folge von Records	28
2. Konkatenation zu σ	20
3. Pointer auf jeden $L = 4$-ten Term	18
4. Front Coding	15.5
5. Minimal Perfect Hashing	13

Indexaufbau

Günstigstes Verfahren zum Aufbau invertierter Listen ist abhängig von der Größe der Dokumentenkollektion.
Naives Vorgehen konstruiert für Dokumente $\mathcal{D} = (D_1, \ldots, D_N)$ und Termfolge $T = (t_1, \ldots, t_M)$ explizit die Term-Dokumenten-Matrix

$$(\mathrm{tf}(t_j, i))_{i,j}$$

wo (Memo) $\mathrm{tf}(t_j, i) \equiv \#$ Vorkommen von Term t_j in Dokument D_i ist.
Verfahren ist nur bei kleinen DBs praktikabel, da sehr speicheraufwendig!
Beispiel: Die Bibel besteht aus $|T| = 8965$ Termen und $N = 31101$ Teildokumenten (Verse u.ä.). Bei 4 Bytes/Matrixeintrag ergibt dies ca. 250 MB Speicherplatzbedarf.

Kann man **im Speicher indexieren**, verwendet man besser dynamische Strukturen:

- ▷ Terme werden in Suchstruktur abgelegt (Suchbaum, Hashtabelle,...).

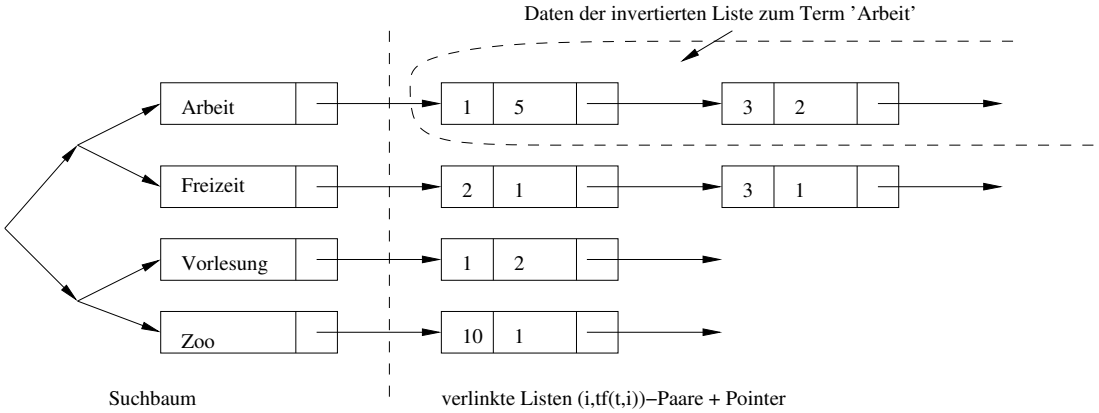
▷ Eintrag zu einem Term in Suchstruktur enthält Pointer auf entsprechende verlinkte Liste.

Schritte zur Indexierung:

1. Erstelle beim Durchlaufen von $\mathcal{D}$ (in aufsteigender Reihenfolge der Dokumentennummern) die verlinkten Listen zu allen Termen. Zähle dazu, wie oft jeder Term in einem Dokument vorkommt.

2. Generiere invertierte Listen durch Auslesen der verlinkten Listen.

Illustration:



Nachteil: Verfahren nicht für Einsatz auf Sekundärspeicher geeignet, da Lesezugriffe zwar im 1. Schritt sequentiell, im 2. Schritt jedoch wahlfrei erfolgen.
Indexierung auf Sekundärspeicher basiert auf dem Prinzip, nur sequentielle Lese- und Schreibzugriffe zu verwenden.

Vorbereitungen:

▷ Verwende Tabelle $S : T \to [1 : |T|]$, die während des Indexierens erstellt wird. $S(t)$ ist die Nummer des Terms t im Index.

▷ Verwende lineare Ordnung

$$(t, d) < (t', d') \Leftrightarrow [t < t' \vee (t = t' \wedge d < d')]$$

auf $[1 : |T|] \times [1 : N]$.

Grobvorgehen zur Indexierung:

1. Durchlaufe für $i \in [1 : N]$ Dokument D_i. Für jeden Term $t \in T$ mit $\mathrm{tf}(t, i) > 0$ füge

$$(S(t), i, \mathrm{tf}(t, i))$$

in Temporärdatei ein.

2. Hat Hauptspeicher Größe H und Temporärdatei Größe G, teile Temporärdatei in $\lceil G/H \rceil$ Stücke und sortiere jedes im Hauptspeicher in situ (z.B. mit Quicksort) gemäß $<$ (die tf-Gewichte bleiben vernachlässigt). Es resultiert eine Temporärdatei aus $\lceil G/H \rceil$ vorsortierten Teilstücken.

3. Verwende externes Mergesort, um die Temporärdatei vollständig bzgl. $<$ zu sortieren.

4. Lese die Temporärdatei sequentiell und erstelle dabei die invertierten Listen (alle Elemente sind wegen der Wahl von $<$ bereits korrekt sortiert).

Nachteil: Das Mergen verursacht hier eine Verdopplung des externen Speicherplatzbedarfs (letzter Merge-Schritt).
Ausweg: Geeignete Kompression der Temporärdatei kann helfen!
Beispielsweise ist zu erwarten, dass durch eine Run-Length-Codierung der Term- und Dokumentnummern (differenziell, z.B. mit γ-Codes) einiges eingespart werden kann. (Hier keine weiteren Details!)

1.5.2.2 Signature Files

Idee: Codiere Auftreten von Termen platzsparend in binärer Matrix

$$B := (B_{ij})_{i \in [1:N], j \in [1:M]}$$

mit

$$B_{ij} := \begin{cases} 1 & \text{falls } t_j \in \tau(D_i), \\ 0 & \text{sonst.} \end{cases}$$

B nennt man *Bitmap* der Dokumentenkollektion (zur gegebenen Termfolge).

Beispiel: Betrachten extrahierte Termmengen

$$\tau(D_1) = \{\text{Clausen, Kurth, Retrieval}\}$$

und

$$\tau(D_2) = \{\text{Kurth, Audio}\}.$$

Dies liefert (bei lexikographischer Ordnung der Terme) die Bitmap B:

Audio	Clausen	Kurth	Retrieval	
0	1	1	1	D_1
1	0	1	0	D_2

$\diamond$

Sei $B(t_j) := (B_{ij})_{i\in[1:N]}$ die Spalte in B zum Term t_j. Definiert man für eine boolesche Anfrage $q \in Q(T)$

$$\mathcal{B}_\mathcal{D}(q) := \begin{cases} B(t_j) & \text{falls } q = t_j \in T, \\ \mathcal{B}_\mathcal{D}(p) \vee \mathcal{B}_\mathcal{D}(r) & \text{falls } q = (p \vee r), \\ \mathcal{B}_\mathcal{D}(p) \wedge \mathcal{B}_\mathcal{D}(r) & \text{falls } q = (p \wedge r), \\ \neg\mathcal{B}_\mathcal{D}(p) & \text{falls } q = (\neg p), \end{cases}$$

wobei die logischen Operatoren $\vee, \wedge$ und $\neg$ auf Vektoren aus $\{0,1\}^N$ arbeiten, und bezeichnet für $x \in \{0,1\}^N$ mit

$$\delta(x) := \{i \in [1:N] \mid x_i = 1\}$$

die Einsstellen von x, dann gilt (Übung!)

$$B_\mathcal{D}(q) = \delta(\mathcal{B}_\mathcal{D}(q)).$$

Die Bitmap eines Dokuments D kann mit Hilfe einer Abbildung

$$\begin{aligned} b : T &\to \{0,1\}^{|T|} \\ t_i &\mapsto e_i, \end{aligned}$$

wobei e_i der i-te Einheitsvektor $(0^{i-1}, 1, 0^{M-i})$ ist, konstruiert werden:

$$b(D) := \bigvee_{t\in\tau(D)} b(t)$$

ist die Bitmap des Dokuments D und es gilt

$$B = \begin{pmatrix} b(D_1) \\ \vdots \\ b(D_N) \end{pmatrix}.$$

Signaturen stellen eine Verallgemeinerung dieses Prinzips dahingehend dar, dass nun beliebige Abbildungen

$$s : T \to \{0,1\}^L$$

zugelassen werden und die *Signaturbreite* $L \in \mathbb{N}$ beliebig wählbar ist.
Um eine kompakte Darstellung zu erhalten, wählt man natürlich i.a. $L < |T|$.
Weiterhin fordern wir, dass kein Term auf den Nullvektor 0^L abgebildet wird.
In Analogie zum obigen erhält man die Signatur eines Dokuments D durch Übereinanderlegen

$$s(D) := \bigvee_{t\in\tau(D)} s(t) \tag{1.2}$$

der Termsignaturen. Diese Art von Signaturbildung nennt man auch *Superimposed Coding*.

1.41 Beispiel Wir betrachten die Signaturfunktion s:

$$\begin{array}{lcl} \text{Audio} & \mapsto & 001 \\ \text{Clausen} & \mapsto & 010 \\ \text{Kurth} & \mapsto & 100 \\ \text{Retrieval} & \mapsto & 101 \end{array}$$

Setzt man als Signaturmatrix

$$S := \begin{pmatrix} s(D_1) \\ \vdots \\ s(D_N) \end{pmatrix}, \tag{1.3}$$

erhält man für die Dokumente mit Termmengen

$$\tau(D_1) = \{\text{Clausen, Kurth, Retrieval}\}$$

und

$$\tau(D_2) = \{\text{Kurth, Audio}\}.$$

aus dem vorherigen Beispiel die Signaturmatrix

$$S = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

Problem beim booleschen Retrieval mit Signaturen:
I.a. ist die Fortsetzung von $s : T \to \{0,1\}^L$ auf Dokumente *nicht injektiv* (b war injektiv!).

- Für die Daten des Beispiels folgt z.B.

  $$s(\text{„Kurth, Audio“}) = 101 = s(\text{„Retrieval“}),$$

  d.h. eine boolesche Suche nach „Kurth $\wedge$ Audio“ anhand der Signatur s liefert auch Dokumente, die nur den Term „Retrieval“ enthalten und die gesuchten Terme nicht. Solche Art von Treffern nennt man auch *false matches* oder *false positives*.

- Führt man eine Suche „$\neg$ Retrieval“ wie bisher aus, indem man

  1. alle Dokumente bestimmt, in denen die Signaturbits von Retrieval (101) gesetzt sind — dies sind die Dokumente $\{1,2\}$ — und
  2. das Komplement dieser Dokumentenmenge bildet, $[1:2]\backslash\{1,2\} = \emptyset$,

  erhält man eine leere Treffermenge. Zu dieser Anfrage ist jedoch Dokument D_2 ein Treffer. Solche fehlenden Treffer nennt man auch *false negatives*.

Diese Probleme erfordern stets eine Nachbearbeitung der Treffermenge. Zur Vermeidung von false negatives betrachten wir hier nur noch *rein konjunktive* Anfragen. Vorgehen zur Anfragebearbeitung:

1. Ermittlung von Trefferkandidaten,
2. Elimination von false positives.

Ermittlung von Trefferkandidaten

Idee: Ist $q = t_1 \wedge t_2 \wedge \ldots \wedge t_k$, bestimme

$$s(q) := \bigvee_{i=1}^{k} s(t_i).$$

Damit D_i Treffer ist, müssen notwendigerweise alle Bits, die in $s(q)$ gesetzt sind, auch in $s(D_i)$ gesetzt sein. Bezeichnet $\Rightarrow$ die logische (komponentenweise) Implikation, können wir diese Bedingung auch als

$$(s(q) \Rightarrow s(D_i)) = 1^L$$

oder äquivalent

$$(\neg s(q) \vee s(D_i)) = 1^L$$

schreiben.

1.42 Beispiel Betrachten $q =$„Kurth $\wedge$ Audio“ mit obiger Signatur (also $L = 3$ und $s(q) = 101, s(D_1) = 111, s(D_2) = 101$), dann gelten

$$\neg s(q) \vee s(D_1) = 111$$

und

$$\neg s(q) \vee s(D_2) = 111,$$

d.h. beide Dokumente sind Trefferkandidaten.

Dies liefert als naive Möglichkeit zur Bestimmung der Treffermenge:

$$\text{Teste } \forall i \in [1:N], \text{ ob } (\neg s(q) \vee s(D_i)) = 1^L.$$

Effizienter ist i.a. folgendes Vorgehen: Man bezeichnet mit $S(j) := (S_{ij})_{i\in[1:N]}$ die j-te Spalte der Signaturmatrix S (siehe (1.3)).
Nun betrachtet man die Einsstellen $\delta(s(q))$ der Anfragesignatur und bildet

$$\bigwedge_{j\in\delta(s(q))} S(j) =: \mathcal{S}_{\mathcal{D}}(q).$$

Dann gilt (Übung!)

$$B_{\mathcal{D}}(q) \subseteq \delta(\mathcal{S}_{\mathcal{D}}(q)),$$

d.h. $\delta(\mathcal{S}_{\mathcal{D}}(q))$ enthält alle Treffer und möglicherweise false positives.

Bemerkung: In Analogie zu oben ($\wedge$ über Terme in der Anfrage) wurde hier nur mit den Einsstellen der Signatur ($\delta(s(q))$) gearbeitet, also auch nur diese Spalten der Matrix S verwendet. Dies kann man ausnutzen, indem man die Spalten von S zusammenhängend auf Sekundärspeicher ablegt. Man spricht dann von *Bitsliced Signatures*.

1.43 Beispiel

$$S = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

wie oben und $q =$„Kurth $\wedge$ Audio“.
$s(q) = 101$, also $\delta(s(q)) = \{1, 3\}$. Dann ist

$$\bigwedge_{j \in \delta(s(q))} S(j) = S(1) \wedge S(3) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \wedge \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \mathcal{S}_{\mathcal{D}}(q)$$

und $\delta(\mathcal{S}_{\mathcal{D}}(q)) = \{1, 2\}$.

Verschiedenes zu Signaturen:

- ▷ Beim *Disjoint Coding* wird die Signatur einer Term*folge* nicht durch Überlagerung von Einzelsignaturen (wie in (1.2)) gewonnen, sondern durch Konkatenation:

  Ist $D =: d_1, \ldots, d_\ell$ Wortfolge, dann bildet man die Folge

  $$s(\tau(d_1)), \ldots, s(\tau(d_\ell)).$$

- ▷ *Block Superimposed Coding* kombiniert Disjoint Coding und Superimposed Coding:

  1. Für aufeinanderfolgende Blöcke von K Termen wird mittels Superimposed Coding eine Signatur gebildet, und
  2. die Codes der so gewonnenen Blöcke werden konkateniert.

- ▷ Meist wählt man die Signaturbreite L unter Annahmen zur Termverteilung und Bedingungen an das Verhältnis

  $$|B_{\mathcal{D}}(q)| : |\delta(\mathcal{S}_{\mathcal{D}}(q))|$$

  von Matches zu (Matches + false positives).

- ▷ Die Organisation von Signaturen auf Sekundärspeicher kann (neben der Bitslice-Methode (vertikal)) auch signaturweise (horizontal) erfolgen. Hier sind verschiedene Organisationsmöglichkeiten in Speicherblöcke möglich, z.B. B-Bäume. Die Signatur wird hierbei jeweils als Zugriffs-„Schlüssel“ verwendet.

- ▷ Signaturen sind historisch vor den invertierten Listen zur platzsparenden Indexierung verwendet worden. Meist liefern invertierte Listen (+ Kompression) jedoch bessere Performanz.

- ▷ Es sind noch andere Abstandsmaße zwischen Signaturen denkbar, z.B. der Hamming-Abstand

  $$d_H(x, y) := \#\{i \mid x_i \neq y_i\}$$

  für Signaturen $x, y \in \{0, 1\}^L$ oder die Anzahl der Einsstellen in der Anfragesignatur x, die in der Dokumentensignatur y nicht vorkommen,

  $$d_c(x, y) := \#\{i \mid x_i = 1 \wedge y_i = 0\}.$$

  d_c heißt *Cover Distance*.

1.6 Literatur

Eine ausführliche Einführung in das klassische Information Retrieval bietet das Lehrbuch [2]. Grundlegende Algorithmen und Datenstrukturen im Information Retrieval finden sich im Sammelband [3].
Indexierungs-, Such- und Kompressionstechniken für die Volltextsuche werden im Lehrbuch [4] behandelt. Algorithmen zur zeichenkettenbasierten Suche finden sich im Lehrbuch [5]. Editierdistanzen, String-Alignment und Techniken des Dynamic Time Warping (DTW) werden ausführlich im Kontext von Algorithmen aus dem Bereich Bioinformatik im deutschsprachigen Lehrbuch [6] besprochen.

Kapitel 2

Web-Retrieval

2.1 Einleitung und Modellierung

Das WWW hat sich seit seinem Entstehen Anfang der 90er Jahre zu einer der meistgenutzten Informationsquellen entwickelt.

Datenbestand:

- WWW-Seiten, meist untereinander verlinkt (Sonderstellung → Navigation)
- Beliebige digitale Dokumente, z.B. Text, Audio, Video

Charakteristika des WWW-Datenbestands im Vergleich zu klassischen Text-Datenbeständen:

	Text-Datenbestand	**WWW-Datenbestand**
allg. Struktur	einzelne, kontrollierte Datenbestände und Kollektionen, die untereinander meist nicht verlinkt sind	globaler, unkontrollierter (unmoderierter) Datenbestand aus untereinander verlinkten Dokumenten. Es existieren Mechanismen zur Abschottung privater Bereiche (→ *Deep Web*)
Größe	klein bis groß (einige MB bis mehrere GB)	sehr groß (allein $>$ 200 GB Text) Durch Multimediadaten potentiell viel größer.
Dokumente	Art und Struktur relativ homogen innerhalb einzelner Datenbestände	Art und Struktur heterogen (digitale Dokumente verschiedener Formate)

Dokumentbeziehungen	Oft keine evtl. in übergeordneter DB verwaltet evtl. spezialisiert, z.B. Literaturquellen	Verlinkung von HTML-Seiten untereinander sowie Links von HTML-Seiten auf Multimediadokumente sind essentiell für das WWW
Datenqualität	oft gut gepflegt, annotiert und validiert	sehr variabel mit vielgestaltigen Fehlern
Treffer zu kurzen Termanfragen	relativ wenige, gute ($\sim k \cdot 100$)	sehr viele, davon viele unerwünschte (oft $\geq n \cdot 100.000$)

Aufgabenstellungen des Web-Retrievals:

- Lokalisation derjenigen Webseiten (in absteigender Wichtigkeit), die am besten zu einer gegebenen (termbasierten) Anfrage passen, z.B. Suche nach Bildern bei Google.
- Lokalisation derjenigen (verlinkten) Multimediadokumente, die am besten zu einer gegebenen (termbasierten) Anfrage passen.
- Ermittlung aller Webseiten, die *ähnlich* zu einer gegebenen Webseite u sind.

  Beispiele:

  - u sei Seite eines Computerherstellers: finde andere Webseiten von Computerherstellern.
  - Seite u enthalte Informationen zu einer bestimmten Erkrankung. Finde Seiten, die Behandlungsmethoden dieser Krankheit beschreiben.

  Achtung: Ähnlichkeit ist nicht genau spezifiziert!
- Ermittlung der kompetentesten Webseiten zu einem bestimmten Thema. ($\rightarrow$ Authorities)
- Ermittlung geeigneter Überblicks- bzw. Einstiegsseiten zu einem Thema. ($\rightarrow$ Hubs)

Inhalt des Web-Retrievals (grob):

Verbinde „klassische“ deterministische Retrievalmethoden und geeignete Heuristiken, um unter Ausnutzung der Struktur des Webs (und der Webdokumente) obige Anfragen möglichst gut beantworten zu können.

Was kann man beim Web-Retrieval ausnutzen?

- Dokumentinterne Annotationen durch Markup-Sprachen
  Beispiele:
  - Text-Hervorhebungen (Fett, kursiv, Großschrift,...)
  - explizite Gruppierung von Textbereichen
  - Markierung von Begriffen (z.B. durch HTML-Tags)
- Dokumentübergreifende Annotationen und Verlinkung ($\rightarrow$ Linkstruktur und Statistiken hierzu)
- Dokumentenzugriffsstatistiken
  - Anzahl Zugriffe auf ein Dokument bzw. einen Server
  - Benutzerverhalten auf Webseiten, insbesondere bei der Benutzung von Suchmaschinen

Modellierung des WWW und Notationen

Wir modellieren das WWW als gerichteten Graphen $G = (V, E)$, bestehend aus einer endlichen Menge V von Knoten ($\equiv$ Webseiten) und $E \subseteq V \times V$ von Kanten ($\equiv$ Links). Zu $W \subseteq V$ bezeichnet

$$G|_W := (W, E|_W) := (W, W^2 \cap E)$$

den durch W induzierten Teilgraphen von G. Zu $v \in V$ ist

$$i(v) := \{w \in V \mid (w, v) \in E\}$$

die Menge der *Vorgänger-* oder *Elternknoten*. Entsprechend bezeichnet

$$o(v) := \{w \in V \mid (v, w) \in E\}$$

die *Nachfolger-* oder *Kindknoten* von v. Die Kardinalitäten dieser Mengen bezeichnen wir mit $i_v := |i(v)|$ und $o_v := |o(v)|$.
Manchmal interessieren wir uns für die Zuordnung von Webseiten zu Host-Rechnern (oder Domänen — im folgenden sprechen wir vereinfachend nur von Hosts). Sei $\mathcal{H}$ eine endliche Menge von Host-IDs, dann bezeichne

$$H : V \rightarrow \mathcal{H}$$

die Zuordnung, die einer Webseite $v \in V$ ihre Host-ID $H(v) \in \mathcal{H}$ zuordnet. Zu $\mu \in \mathcal{H}$ bezeichnet dann $H^{-1}(\mu) \subseteq V$ die Menge aller Webseiten des Hostes μ.
Die Zuordnung von Webseiten zu bestimmten *Hosts* wird hier rein logisch, unabhängig von physikalischen Hostrechnern, Domänen oder (Sub-) Netzen vorgenommen. Die Motivation liegt hierbei darin, bestimmte logisch zueinander gehörige Verbünde von Webseiten, wie etwa das Webangebot des Instituts für Informatik (mehrere Webserver, mehrere Domains), gruppieren zu können. Mit Hilfe solch

logischer Gruppierungen können dann Verweise von Webseiten einer Gruppe untereinander besser modelliert und auf diese Weise unerwünscht häufige Verweise auf bestimmte Webseiten vermieden werden. Wie eine solche Host-Gruppierung in der Praxis realisiert wird, bleibt der jeweiligen Anwendung überlassen und wird hier nicht weiter diskutiert.

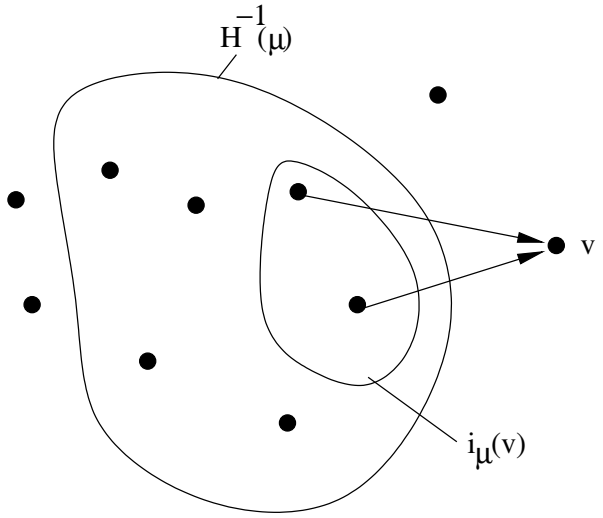


Abbildung 2.1: Menge $i_\mu(v)$ aller Webseiten auf einem Host-Rechner $\mu \in \mathcal{H}$, die einen Link auf eine Webseite v besitzen.

Die Menge aller Webseiten auf einem Host-Rechner $\mu \in \mathcal{H}$, die einen Link auf eine Webseite v besitzen, wird mit

$$i_\mu(v) := \{w \in H^{-1}(\mu) \mid (w, v) \in E\}$$

bezeichnet (siehe Abb. 2.1). Die Menge aller Webseiten auf einem Hostrechner $\mu \in \mathcal{H}$, auf die ein Link von einer Webseite v zeigt, wird entsprechend mit

$$o_\mu(v) := \{w \in H^{-1}(\mu) \mid (v, w) \in E\}$$

bezeichnet (siehe Abb. 2.2).
Sei $G = (V, E)$ der WWW-Graph und $(v_1, \ldots, v_n)$ eine beliebige aber feste Nummerierung der Webseiten von V. Die Matrix $A(G) := (a_{ij})_{1 \le i,j \le n}$ mit

$$a_{i,j} := \begin{cases} 1 & \text{falls } (v_i, v_j) \in E, \\ 0 & \text{sonst,} \end{cases}$$

heißt *Adjazenzmatrix* zu G (bzgl. der gegebenen Ordnung). Ist G aus dem Zusammenhang klar, schreiben wir einfach $A = A(G)$.

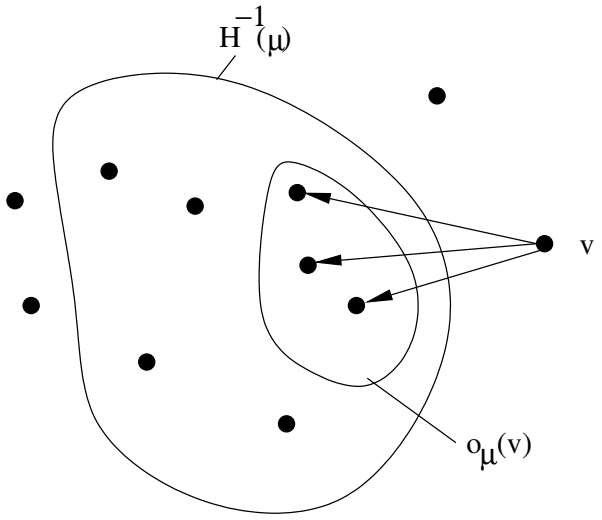


Abbildung 2.2: Menge $o_\mu(v)$ aller Webseiten auf einem Hostrechner $\mu \in \mathcal{H}$, auf die ein Link von einer Webseite v zeigt.

2.1 Bemerkung

- Links innerhalb einer Webseite v können durch Kanten (v, v) modelliert werden.
- Eine genaue Positionierung von Links auf einer Webseite ($\rightarrow$ Anker) wird hier nicht modelliert.
- Möchte man Mehrfachlinks zwischen zwei Webseiten v und w modellieren, wird E zu einer Multimenge, in der eine Kante (v, w) mehrfach vorkommen kann. Dies wird formalisiert durch eine Abbildung

$$E : V \times V \rightarrow \mathbb{N}_0,$$

die einer Kante (e, f) die Häufigkeit ihres Vorkommens im Graphen zuordnet. Die Adjazenzmatrix (a_{ij}) ist in diesem Falle gegeben durch $a_{ij} := E(v_i, v_j)$.

$\diamond$

2.2 Die „WWW-Datenbank“

Da es keine „zentrale Verwaltung“ des WWW-Datenbestands gibt, sind Aussagen über Größen wie

- Datenvolumen

- Anzahl Webseiten
- Anzahl Webserver
- Abdeckung der Webseiten durch Suchmaschinen

schwierig. Man behilft sich hier momentan mit vereinfachenden Schätzungen. Nimmt man z.B. an, dass es im IP-Adressraum mit 2^{32} Adressen S Webserver gibt, deren Adressen *gleichmäßig* auf den Adressraum verteilt sind, so kann man versuchen, aus einer Stichprobe betrachteter Adressen hochzurechnen. Eine gleichmäßige Verteilung der Webserver kann z.B. bedeuten, dass für eine hinreichend große Menge $W \subseteq [0 : 2^{32} - 1]$ von IP-Adressen für die Menge

$$S(W) := \{w \in W \mid w \text{ ist IP-Adresse eines Webservers}\}$$

stets $|S(W)|/|W| \approx c$ für eine Konstante $c > 0$ gilt. Beobachtet man in der Stichprobe aus N IP-Adressen M Webserver, schätzt man nun linear via $c \approx M/N$ die Gesamtanzahl der Webserver zu $S \approx 2^{32}M/N$.
Meßwerte aus dem Jahre 1999 ergeben z.B.

- $M/N \approx 1/269$
- $S \approx 16$ Millionen

Achtung: Bei solchen Messungen werden i.a. auch IP-Adressen als Serveradressen gezählt, die das HTTP-Protokoll verwenden, obwohl kein Web-Server vorhanden ist (z.B. Drucker, die über HTTP kommunizieren).

Laut *Netcraft Web Server Survey*, `www.netcraft.com`, vom August 2002 gibt es inzwischen ca. 36 Millionen Webserver (63,5 % Apache, 25.39% Microsoft-IIS, alle anderen jeweils unter 2,2%).

Schätzen der Größenverhältnisse von Suchmaschinen

Für den Rest des Kapitels sei das WWW wie im vorherigen Abschnitt beschrieben als Graph $G = (V, E)$ modelliert. Wir nehmen an, eine Suchmaschine a hat eine Teilmenge $A \subseteq V$ von Webseiten indexiert, eine Suchmaschine b eine andere Teilmenge $B \subseteq V$. Gesucht ist das Verhältnis $|A|/|B|$. Da wir nicht auf ganz A bzw. B zugreifen können, wählen wir Stichproben $A' \subseteq A$ bzw. $B' \subseteq B$. Wir ermitteln dann, welcher Anteil der Stichprobe B' auch in A liegt (Seiten der Suchmaschine b, die auch a indexiert hat),

$$\frac{|A \cap B'|}{|B'|} =: P(A|B').$$

(Unter Annahme einer Gleichverteilung P auf der Menge V ist die l.S. tatsächlich die bedingte W'keit $P(A|B')$.) Unter der Annahme, dass dieser Anteil auf ganz B konstant ist, können wir linear

$$\frac{|A \cap B|}{|B|} \approx \frac{|A \cap B'|}{|B'|}$$

approximieren. Analog bestimmen wir

$$\frac{|A' \cap B|}{|A'|} =: P(B|A'),$$

den Anteil der Webseiten in B an A' und approximieren

$$\frac{|A \cap B|}{|A|} \approx \frac{|A' \cap B|}{|A'|}.$$

Wir erhalten dann das gesuchte Verhältnis durch

$$\frac{|A|}{|B|} = \frac{\frac{|A \cap B|}{|B|}}{\frac{|A \cap B|}{|A|}} \approx \frac{P(A|B')}{P(B|A')}.$$

2.2 Beispiel Im November 1997 wurden die Größen verschiedener Suchmaschinen zueinander geschätzt. Ein Auszug hieraus:

A indexiert	B von	$A' \subseteq A : \frac{\lvert A' \cap B\rvert}{\lvert A'\rvert} \cdot 100$	Schätzung $\frac{\lvert A\rvert}{\lvert B\rvert}$
Alta Vista	Exite	15%	3,23
	HotBot	39%	1,28
	InfoSeek	15%	3,32

◇

Schätzen der Anzahl verfügbarer WWW-Seiten

Modelliert man die Auswahl der zu indexierenden Seiten der Suchmaschinen durch die Auswahl von Mengen A und B unter einer Gleichverteilung P auf V und nimmt zusätzlich stochastische Unabhängigkeit an, $P(A \cap B) = P(A)P(B)$, so folgt

$$\begin{aligned} & \frac{|A \cap B|}{|V|} = P(A \cap B) = P(A)P(B) = \frac{|A||B|}{|V|^2} \\ \Leftrightarrow \quad & \\ & |V| = \frac{|A||B|}{|A \cap B|}. \end{aligned}$$

Kann man $|A|$ oder $|B|$ bestimmen (manchmal sind Infos zu Suchmaschinen öffentlich bekannt), so kann man versuchen, durch Stichproben $A' \subseteq A$

$$|A \cap B| \approx |A' \cap B| \cdot \frac{|A|}{|A'|}$$

zu approximieren. Mittelung über verschiedene Suchmaschinen und Stichproben kann zur Schätzung der Anzahl $|V|$ an Webseiten dienen.

Fragen aus Sicht der Praxis:

1. Wie wählt man $A' \subseteq_R A$, wenn man auf A nur indirekt (über die Anfrageschnittstelle der Suchmaschine) Zugriff hat?

   Schreibweise: Mit $A \subseteq_R B$ bezeichnen wir die zufällige Auswahl einer Teilmenge A aus B (analog $a \in_R B$ für die zufällige Auswahl eines Elements a aus A). Oft stellen wir in diesem Zusammenhang eine Anforderung an die Größe $|A|$. Wird nichts anderes erwähnt, legen wir bei der Auswahl eine Gleichverteilung zu Grunde.

2. Wie bestimmt man — unter analogen Gesichtspunkten — $A' \cap B$?

Zu 1.: Man generiert eine zufällige Anfrage an die Suchmaschine. Die Antwort ist eine Teilmenge $\tilde{A} \subseteq A$. Wähle nun $A' \subseteq_R \tilde{A}$.

Zu 2.: Erzeuge Anfrage zu $a \in A'$, die sehr spezifisch ist (lasse nur seltene Terme zu). Ziel: Anfrage zu a an eine zweite Suchmaschine, die B indexiert hat, soll Ergebnis liefern gdw. $a \in B$.

Ein paar Zahlen & Statistiken zum WWW und zu Suchmaschinen

- Entwicklung der Anzahl der Webseiten

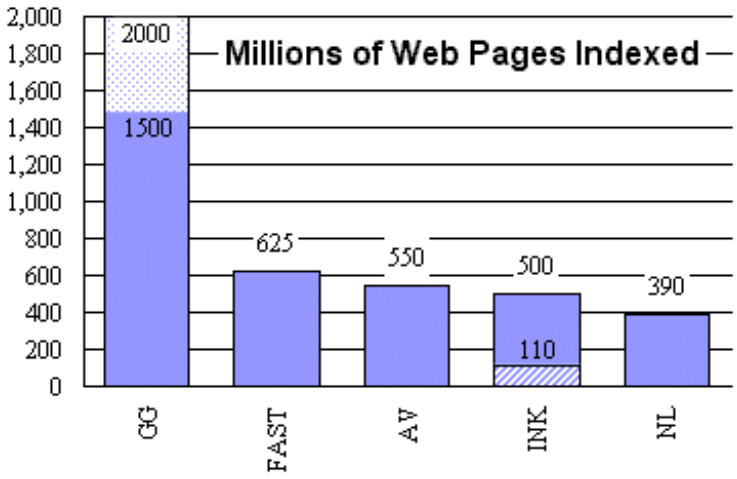
12/97 $>$ 320 **Millionen** Seiten, alle großen Suchmaschinen indexieren zusammen 60 % der Seiten

02/99 **ca. 800 Millionen** Seiten, ca. 14 TB Daten (HTML)

01/00 $>$ 1 **Milliarde** Seiten

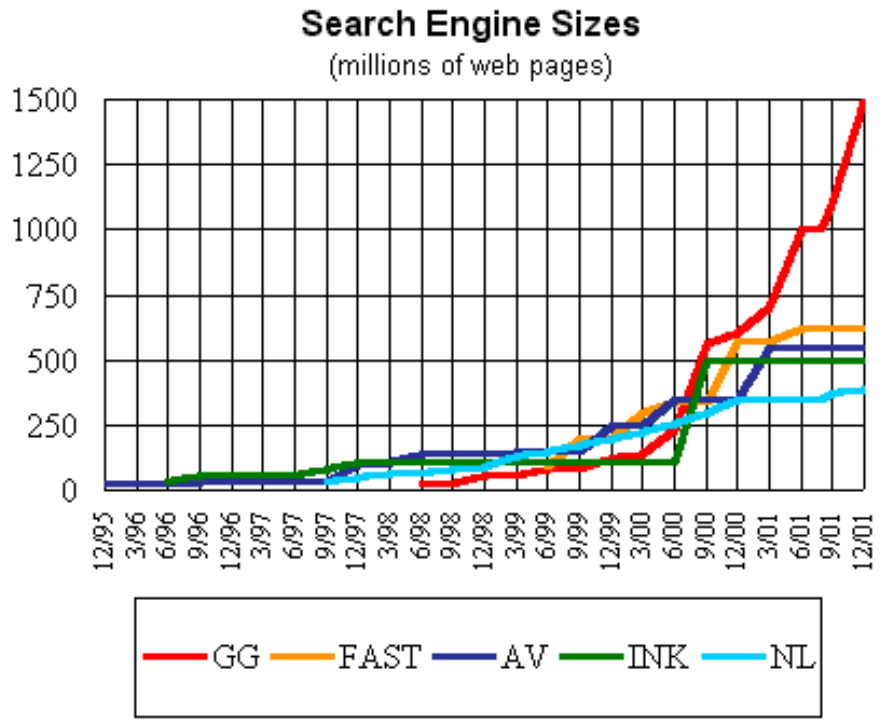
12/00 **3 – 5 Milliarden** Seiten, Google indexiert ca. 30 % der Seiten (einige nur implizit über Links), Datenvolumen des Web ca. 19 Terabyte (HTML), 7500 Terabyte *deep Web* — Web incl. nicht öffentlich zugänglichen und automatisch generierten Seiten.

- Suchmaschinengrößen lt. `www.searchEngineWatch.com` am 11.12.2001.



GG — Google, FAST — Fast, AV — Alta Vista, INK — Inktomi, NL — NorthernLight

- Entwicklung der Suchmaschinengrößen von 1995 – 2001.



- Anfragen an Suchmaschinen pro Tag

Suchmaschine	**Anfragen pro Tag [in Millionen]**	**Datum**
Google	150	05/02
Inktomi	80	08/01
AltaVista	50	03/00
Direct Hit	20	04/01
FAST	12	10/00
Overture (GoTo)	6,5	04/02
Ask Jeeves	4	03/00

2.3 Standard WWW-Suche

Üblicherweise: Boolesches Retrieval oder Vektorraumretrieval anhand von angefragten Termen.

Wesentlich hierbei: Geeignetes *Ranking* der Suchergebnisse, d.h. Ordnen der Suchergebnisse nach ihrer (geschätzten) Relevanz.

In diesem Abschnitt geht es hauptsächlich um das geschickte Ranking anhand von Heuristiken und der (Link-) Struktur des WWW.

2.3.1 Google-Suche

In der Suchmaschine Google werden einige der zur Zeit erfolgreichsten Ranking-Techniken verwendet. Daher besprechen wir hier exemplarisch die Funktionsweise dieser Suchmaschine. Memo: Der Graph $G = (V, E)$ bestehend aus Knoten V (Webseiten) und Kanten E (Links) modelliere das WWW.

Aufbau des Suchindexes

1. Eine Teilmenge $V' \subseteq V$ von WWW-Seiten wird indexiert. Hierzu wird jedes $v \in V'$ geparst und (nach der üblichen Termextraktion) in eine Folge $(v_1, \ldots, v_n)$ von Indexobjekten übersetzt. Hierbei ist

$$v_i \in T \times D \times P \times F \times A \times K =: \mathcal{I},$$

wobei

$T \subseteq \Sigma^*$ eine Menge gültiger Terme über einem Alphabet Σ,

$D \subseteq \mathbb{N}$ eine Menge von Dokumenten-IDs,

$P \subseteq \mathbb{N}$ Positionen innerhalb eines Dokuments,

$F \subseteq \mathbb{Z}$ eine Menge von Inkrementen und Dekrementen der aktuellen Fontgröße (größer/kleiner),

$A = \{\text{normal, bold, italic,}\ldots\}$ Fontattribute

$K = \{\text{URL, Titel, Meta-Tag, Linktext, Sonstiges,}\ldots\}$ Kontextattribute

sind. Ist $x \in \mathcal{I}$, so bezeichnen wir mit $T(x), D(x), P(x), F(x), A(x), K(x)$ die Projektionen auf die einzelnen vorgenannten Einträge von x.

2.3 Beispiel Das Dokument

```
<HTML>
<TITEL>
Demonstration
</TITEL>
<BODY>
<B><FONT SIZE=+1> Indexierung </FONT></B>
</BODY>
</HTML>
```

könnte zu einer Folge (v_1, v_2) mit

$$v_1 = (\text{Demonstration}, 1, 1, 0, \text{normal}, \text{Titel}),$$
$$v_2 = (\text{Indexierung}, 1, 2, 1, \text{bold}, \text{Sonstiges}),$$

führen. ◇

2. Die indexierte Teilmenge V' wird angereichert um die Webseiten

$$V'' := \{v \in V \setminus V' \mid \exists v' \in V' : (v', v) \in E\}.$$

Dies sind alle von den bisher indexierten Webseiten aus V' referenzierten Seiten, die nicht selbst in V' liegen. Die Seiten aus V'' wurden sämtlich *nicht* von Google besucht. Jedoch werden alle Texte innerhalb aller Links auf eine Seite $v \in V''$ aus V' heraus zusammengestellt und wie oben indexiert. (Hierbei wird eine neue ID für die Seite $v \in V''$ erstellt, und die indexierten Terme dieser ID zugewiesen.)

→ Google indexiert daher mehr Seiten, als wirklich besucht (gecrawlt) wurden.

3. Zur Bestimmung des von Google im folgenden verwendeten Subgraphen des WWW bilden wir zunächst den Subgraphen

$$(V_g, E^*) := G|_{V' \cup V''}$$

bezüglich der Seiten V' und V'' und entfernen hieraus alle Links, die von Seiten in V'' ausgehen,

$$E_g := E^* \setminus \{(v, w) \mid v \in V''\}.$$

(Bemerkung: $E_g = (V' \times (V' \cup V'')) \cap E$.) Der von Google verwendete Subgraph ist dann $G_g := (V_g, E_g)$.

Ranking I: Proximity der Terme

Eine Anfrage — nach Termextraktion und Verwerfen von Stopwörtern — ist eine Folge von Termen $q = (q_1, \ldots, q_m) \in T^m$.

Idee: Bei jeder Webseite, die ein Trefferkandidat zu einer Anfrage ist wird untersucht, ob benachbarte Terme der Anfrage auch auf dieser Webseite benachbart sind. Abb. 2.3. illustriert das Konzept von Term-Nachbarschaft.
Dies führt zu einer Gewichtung des Trefferkandidaten $v = (v_1, \ldots, v_k) \in \mathcal{I}^k$ abhängig von der Nähe der Terme aus q in v. Für ein Paar $p, p' \in T$ von Anfragetermen definieren wir die Menge

$$H(p, p', v) := \{(h, \ell) \mid \exists v_r : T(v_\ell) = p \wedge T(v_r) = p' : h = P(v_r) - P(v_\ell)\}$$

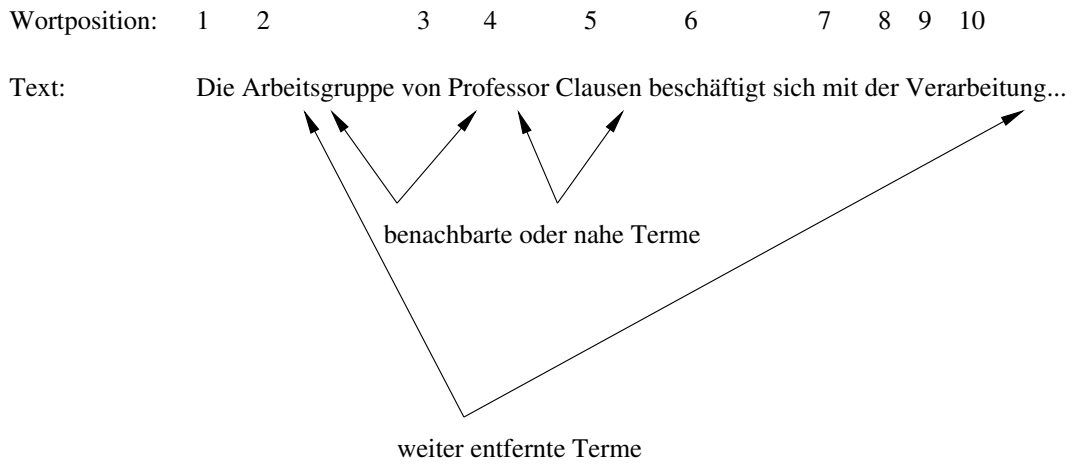


Abbildung 2.3: Benachbarte und voneinander entfernte Terme.

aller Vorkommen beider Terme in einer Webseite $v =: (v_1, \ldots, v_k) \in \mathcal{I}^k$. Eine Quantisierung der Abstände h auf 10 disjunkte Abstandsklassen mit anschließender Histogrammbildung liefert einen Gewichtsvektor

$$(H_1(v), \ldots, H_{10}(v))$$

wobei

$$H_j(v) = |\{(h, \ell) \in H(p, p', v) \mid |h| \in I_j\}|$$

und

$$\mathbb{N} =: I_1 \sqcup \ldots \sqcup I_{10}$$

eine geeignete Partitionierung ($\rightarrow$ Quantisierung) in Abstandsklassen ist, z.B.

$$[1] \sqcup [2] \sqcup [3:4] \sqcup [5:7] \sqcup [8:11] \sqcup [12:15] \sqcup [16:25] \sqcup [26:50] \sqcup [51:150] \sqcup [151:\infty).$$

Multiplikation mit dem Vektor $w := (1, 0.9, \ldots, 0.1)$ zur Gewichtung der Abstandsklassen liefert bezüglich der Anfrageterme p, p' das Gewicht

$$w_{p,p',v} := (H_1(v), \ldots, H_{10}(v))w^\top$$

des Dokuments v. Zur Einbeziehung aller Anfrageterme kann dann z.B. das Gesamtgewicht

$$w^1_{v,q} := \sum_{k=1}^{m-1} w_{q_k, q_{k+1}, v}$$

des Dokuments v zur Anfrage $q = (q_1, \ldots, q_m) \in T^m$ berechnet werden.
Abb. 2.4 zeigt beispielhaft eine Einteilung in 10 Proximity-Klassen.

Klasse	Nähe der Terme	Beispielhäufigkeit
1	nebeneinander	2
2	benachbart	0
3	⋮	1
4	⋮	4
5	weitere Umgebung	1
6	⋮	0
7	⋮	1
8	entfernt	1
9	⋮	1
10	sehr weit entfernt	5

Abbildung 2.4: Statistik der Abstands- oder Proximity-Klassen irgendeines Beispieldokuments. Man erhält mit obigem w das Gewicht 7,6.

Ranking II: Kontextinformation der Terme

Idee: Verwende Informationen über den Kontext der Terme (z.B., ob sie in einem Link vorkommen, oder fett gedruckt sind) zur weiteren Gewichtung.

2.4 Beispiel Man kann etwa unter Verwendung folgender Parameterwahlen eine auf dem Termkontext basierende Gewichtung eines Beispieldokuments durchführen:

Kontext, z.B. HTML-Tag	**Häufigkeit im Beispieldokument**	**Häufigkeits-obergrenze**	**Gewichtungs-faktor**
Titel	1	1	5.0
Meta	1	1	4.5
P	672	100	0.05
B	14	12	3.0
I	7	7	2.7
H1	1	1	4.0
H2	12	12	3.6
⋮	⋮	⋮	⋮
Linktext	892	100	7.5

Im Beispieldokument kommen die Suchterme z.B. 14-mal fettgedruckt (Zeile B) vor. Zur Vermeidung übermäßiger Beeinflussungen von außen wird jedoch eine Häufigkeitsobergrenze von 12 festgesetzt. Die nach Beschränkung auf diesen Maximalwert erhaltene Häufigkeit wird mit dem Gewichtungsfaktor 3.0 multipliziert. Ein Gesamtgewicht erhält man durch Aufsummierung aller gewichteten Häufigkeiten. ◇

Formal definiert man für eine Anfrage $q = (q_1, \ldots, q_m) \in T^m$ und ein Dokument $v = (v_1, \ldots, v_n)$ auf den Kontext- und Attributmengen eine Abbildung

$$\begin{aligned} h_{v,q} : K \cup A &\rightarrow \mathbb{N} \\ t &\mapsto \begin{cases} \min\{\sigma(t), |\{v_\ell \mid \exists r : T(v_\ell) = q_r \wedge A(v_\ell) = t\}|\} & \text{falls } t \in A, \\ \min\{\sigma(t), |\{v_\ell \mid \exists r : T(v_\ell) = q_r \wedge K(v_\ell) = t\}|\} & \text{falls } t \in K. \end{cases} \end{aligned}$$

Hier sei $\sigma : K \cup A \rightarrow \mathbb{N}$ eine Funktion, die die Häufigkeitsobergrenze $\sigma(t)$ eines Typs $T \in K \cup A$ angibt. Sei nun analog hierzu $W : K \cup A \rightarrow \mathbb{R}_{\geq 0}$ eine positive Gewichtungsfunktion auf den Kontexttypen, dann definieren wir das kontextbasierte Rankinggewicht des Dokuments v (zur Anfrage q) als

$$w_{v,q}^2 := \sum_{t \in K \cup A} h_{v,q}(t) W(t).$$

Ranking III: PageRank

Idee: Wir betrachten das Websurfen als speziellen stochastischen Prozess (einen sog. *Random Walk*) auf dem Web-Graphen G_g und bestimmen die Wahrscheinlichkeit, dass ein Websurfer auf einer bestimmten Webseite landet. Diese Wahrscheinlichkeit bildet den Rang der Webseite und wird auch *PageRank* genannt (nach einer Arbeit von Brin & Page [7]).

- Ein Websurfer wählt — ausgehend von einer Webseite v als nächste Webseite entweder
  - mit Wahrscheinlichkeit $0 \leq 1 - p \leq 1$ eine der von v aus verlinkten Seiten aus $o(v)$ oder
  - mit Wahrscheinlichkeit p eine beliebige andere Seite des WWW (genauer: aus V_g) per Eingabe einer URL. Hierbei liege eine Gleichverteilung zu Grunde. Somit hat jede Seite aus V_g die Wahrscheinlichkeit $p/|V_g|$.
- Die Anzahl eingehender Links einer Webseite beeinflußt die Wahrscheinlichkeit, dass der Surfer auf dieser Webseite landet. Dies soll ebenfalls modelliert werden. Um zu verhindern, dass — zu Manipulationszwecken — bestimmte Seiten durch sehr viele ausgehende Links den Rang anderer Seiten erhöhen, erhält ein ausgehender Link einer Seite v das Gewicht $1/o_v$.

Aus dieser Motivation macht man für den PageRank folgenden Ansatz. Gesucht ist eine Rang-Funktion $r : V_g \rightarrow \mathbb{R}_{\geq 0}$, so dass die rekursive Beziehung

$$r(v) = \frac{p}{|V_g|} + (1 - p) \sum_{v' \in i(v)} \frac{r(v')}{o_{v'}} \tag{2.1}$$

erfüllt ist. Der erste Term stellt hier die Wahrscheinlichkeit dar, dass der Surfer zufällig auf die Seite springt, während der zweite Term die Wahrscheinlichkeit modelliert, dass der Surfer einem Link einer Vorgängerseite $v' \in i(v)$ folgt. Diese Wahrscheinlichkeit hängt wiederum rekursiv vom Rang $r(v')$ der Vorgängerseite und der Anzahl von v' ausgehender Links ab.
Aufgrund der gegenseitigen Abhängigkeiten (Rekursion & mögliche Zyklen in der Verlinkung) ist es zunächst gar nicht klar, dass r wohldefiniert ist, oder ob überhaupt konsistent Belegungen für Funktionswerte von r existieren. Aufgrund der speziellen Wahlen der Übergangswahrscheinlichkeiten in (2.1) kann man dies jedoch zeigen, indem man des Websurfen als Markov-Prozess modelliert.
Wir erinnern hierzu zunächst an einige Definitionen und Fakten aus der Stochastik.

2.5 Definition Sei $(\Omega, \mathcal{A}, P)$ ein W-Raum, I eine höchstens abzählbar unendliche Zustandsmenge. Eine Familie $(X_t)_{t\in\mathbb{N}_0}$ von Zufallsvariablen

$$X_t : \Omega \to I$$

heißt *stochastischer Prozess*. ◇

2.6 Bemerkung Die Zufallsvariablen induzieren verschiedene W-Verteilungen. Zur Erinnerung aus der elementaren Stochastik: Seien $(\Omega, \mathcal{A}, P)$ ein W-Raum und $(\Omega', \mathcal{A}')$ ein Maßraum, dann heißt $X : \Omega \to \Omega'$ eine $(\mathcal{A}, \mathcal{A}')$-Zufallsvariable genau dann, wenn für alle $A' \in \mathcal{A}' : X^{-1}[A'] \in \mathcal{A}$. In unserem Fall ist $\Omega' = I$, $|I| \leq |\mathbb{N}|$, $\mathcal{A}' = \mathcal{P}(I)$. Dann ist X eine $(\mathcal{A}, \mathcal{A}')$-Zufallsvariable genau dann, wenn für alle $i \in I : X^{-1}[i] \in \mathcal{A}$. In diesem Fall ist $P(X^{-1}[i])$ definiert und wir erhalten z.B. für $t \in \mathbb{N}_0$ Verteilungen

$$P_{X_t} : I \ni i \mapsto P(X_t^{-1}[i]),$$

sowie gemeinsame Verteilungen $P_{X_{t_1},\ldots,X_{t_n}}$ für $\{t_1, \ldots, t_n\} \subseteq \mathbb{N}_0$,

$$I^n \ni (i_1, \ldots, i_n) \mapsto P(X_{t_1}^{-1}[i_1] \cap \ldots \cap X_{t_n}^{-1}[i_n]).$$

Wir verwenden hier die allgemein üblichen suggestiven Schreibweisen

$$P_{X_t}(i) \equiv P(X_t = i) \text{ und } P_{X_{t_1},\ldots,X_{t_n}}(i) \equiv P(X_{t_1} = i_1, \ldots, X_{t_n} = i_n).$$

◇

2.7 Definition Eine *Markov-Kette* ist ein stochastischer Prozess $(X_t)_{t\in\mathbb{N}_0}$, der folgende sogenannte *Markovbedingung* erfüllt:

$$\forall n \in \mathbb{N}_0 \text{ und } i_0, \ldots, i_{n+1} \in I \text{ mit } P(X_0 = i_0, \ldots, X_n = i_n) > 0$$

gilt

$$P(X_{n+1} = i_{n+1} | X_0 = i_0, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n).$$

◇

Interpretation:

- X_t beschreibt den Zustand eines Systems zum Zeitpunkt t. Hier können wir uns darunter z.B. die zum Zeitpunkt t besuchte Webseite vorstellen.
- In unserem Web-Szenario könnte dann z.B. $I := V_g$ die Menge aller von Google indexierten Web-Seiten sein.
- Die Markov-Eigenschaft besagt, dass die Wahrscheinlichkeit, zum Zeitpunkt $n+1$ in einen beliebigen Zustand zu gelangen, nur

  1. vom Zustand zum Zeitpunkt n und
  2. von n

  abhängt.

Spezialisiert man durch Verzicht auf die mögliche Abhängigkeit von $n > 0$, erhält man folgende

2.8 Definition Eine Markov-Kette heißt *homogen*, wenn $\forall n \in \mathbb{N}_0$ und $i, j \in I$

$$P(X_{n+1} = j | X_n = i) = p_{ij}$$

gilt, d.h. die Übergangswahrscheinlichkeiten von Zustand i in Zustand j unabhängig von n sind. ◇

Für eine beliebige aber feste Reihenfolge der Elemente aus I ordnet man einem homogenen Markov-Prozess die Matrix

$$P := (p_{ij})_{(i,j)\in I\times I}$$

zu. Es gelten $\forall i, j : p_{i,j} \geq 0$ sowie

$$\forall i \in I : \sum_{j\in I} p_{ij} = 1.$$

Matrizen mit dieser Eigenschaft nennt man *stochastisch*. Ab jetzt seien alle betrachteten Markov-Prozesse homogen. Das Wahrscheinlichkeitsmaß $\pi := (\pi_i)_{i\in I}$ mit $\pi := (\pi)_{i\in I}$ nennt man *Startverteilung*. Dann gilt

2.9 Lemma

1. $P(X_0 = i_0, \ldots, X_n = i_n) = P(X_0 = i_0) \cdot P(X_1 = i_1 | X_0 = i_0) \cdots P(X_n = i_n | X_{n-1} = i_{n-1})$,
2. $P(X_0 = i_0, \ldots, X_n = i_n) = \pi_{i_0} \cdot p_{i_0 i_1} \cdots p_{i_{n-1} i_n}$.

◇

Beweis: 1. ist klar aus den elementaren Eigenschaften der bedingten Wahrscheinlichkeit. 2. folgt aus 1. □

Für uns ist im folgenden interessant, mit welcher Wahrscheinlichkeit man nach m Schritten aus einem Zustand $i \in I$ in einen Zustand $j \in I$ gelangt,

$$p_{ij}^{(m)} := P(X_{n+m} = j | X_n = i).$$

Man sieht leicht, dass $p_{ij}^{(m)}$ für homogene Markov-Ketten unabhängig von n ist. Weiterhin kann man leicht nachrechnen, dass folgendes Resultat gilt:

2.10 Lemma Für $m \geq 1$ und eine homogene Markov-Kette mit Übergangsmatrix $P = (p_{ij})_{(i,j)\in I\times I}$ gilt

$$(p_{ij}^{(m)})_{(i,j)\in I\times I} = P^m.$$

◇

m-fache Matrixmultiplikation liefert also durch $p_{ij}^{(m)} = (P^m)_{ij}$ die Wahrscheinlichkeit, aus Zustand i in genau m Schritten in den Zustand j zu gelangen.
Das folgende Resultat ist für uns wesentlich. Es besagt, dass die Folge $\mathbb{N}_0 \ni m \mapsto P^m$ unter bestimmten Bedingungen konvergiert.

2.11 Satz $P = (p_{ij}) \in [0,1]^{N\times N}$ sei Übergangsmatrix einer homogenen Markov-Kette, also zeilenstochastisch. Ferner gebe es ein $L \geq 1$, so dass P^L lauter positive Einträge hat. Dann gibt es eine Wahrscheinlichkeitsverteilung $\rho = (\rho_1, \ldots, \rho_N)$, die sog. *Grenzverteilung*, mit

(1) $\lim_{m\to\infty} P^m$ existiert und in jeder Zeile der resultierenden Matrix P^∞ steht die Grenzverteilung ρ.

(2) Die Grenzverteilung ist die eindeutig bestimmte Wahrscheinlichkeitsverteilung mit $\rho P = \rho$.

◇

Beweis. Wir betrachten für festes $j \in [1:N]$ die Folge der Minima und Maxima in den j-ten Spalten der Matrizen $P, P^2, P^3, \ldots$. Dazu sei $P^n =: (p_{ij}^{(n)})$ und

$$m_j^{(n)} := \min_i p_{ij}^{(n)} \quad \text{sowie} \quad M_j^{(n)} := \max_i p_{ij}^{(n)}.$$

Dann ergeben sich aus der allgemein für $(i,j) \in [1:N]^2$ und alle $a, b \in \mathbb{N}$ gültigen Formel

$$p_{ij}^{(a+b)} = \sum_{\ell=1}^{N} p_{i\ell}^{(a)} p_{\ell j}^{(b)}$$

und der Zeilenstochastizität von P folgende Monotoniebeziehungen:

$$m_j^{(n+1)} = \min_i \sum_{\ell=1}^{N} p_{i\ell} p_{\ell j}^{(n)} \geq \sum_{\ell=1}^{N} p_{i\ell} m_j^{(n)} = m_j^{(n)}$$

und analog

$$M_j^{(n+1)} = \max_i \sum_{\ell=1}^{n} p_{i\ell} p_{\ell j}^{(n)} \leq \sum_{\ell} p_{i\ell} M_j^{(n)} = M_j^{(n)}.$$

Also ist die Minima-Folge $(m_j^{(n)})_{n\geq 1}$ monoton steigend, während umgekehrt die Maxima-Folge $(M_j^{(n)})_{n\geq 1}$ monoton fallend ist.

Da es nur endlich viele Paare (i,j) gibt und alle $p_{ij}^{(L)}$ positiv sind, gilt weiter:

$$\exists\, 0 < \delta < 1\, \forall (i,j): \quad 0 < \delta \leq p_{ij}^{(L)} < 1. \tag{2.2}$$

Insbesondere gilt für alle j stets $m_j^{(L)} \geq \delta$. Für Zeilenindizes $h, i \in [1:N]$ seien

$$I^+(h,i) := \{k \mid p_{hk}^{(L)} \geq p_{ik}^{(L)}\} \quad \text{und} \quad I^-(h,i) := [1:N] \setminus I^+(h,i).$$

Damit gilt (nach getrennter Zusammenfassung der positiven bzw. negativen Summanden und wegen $[1:N] = I^+(h,i) \sqcup I^-(h,i)$):

$$\sum_{k\in I^+(h,i)} (p_{hk}^{(L)} - p_{ik}^{(L)}) + \sum_{k\in I^-(h,i)} (p_{hk}^{(L)} - p_{ik}^{(L)}) = 1 - 1 = 0. \tag{2.3}$$

Sind nun für festes n die Indizes (h,i) so gewählt, dass

$$M_j^{(n+L)} = p_{hj}^{(n+L)} \quad \text{und} \quad m_j^{(n+L)} = p_{ij}^{(n+L)},$$

so ist

$$\begin{aligned} M_j^{(n+L)} - m_j^{(n+L)} &= p_{hj}^{(n+L)} - p_{ij}^{(n+L)} = \sum_k (p_{hk}^{(L)} - p_{ik}^{(L)}) p_{kj}^{(n)} \\ &\leq \sum_{k\in I^+(h,i)} (p_{hk}^{(L)} - p_{ik}^{(L)}) M_j^{(n)} + \sum_{k\in I^-(h,i)} (p_{hk}^{(L)} - p_{ik}^{(L)}) m_j^{(n)} \\ &= \sum_{k\in I^+(h,i)} (p_{hk}^{(L)} - p_{ik}^{(L)})(M_j^{(n)} - m_j^{(n)}) \quad \text{wg. (2)} \\ &\leq (1-\delta)(M_j^{(n)} - m_j^{(n)}). \quad \text{(Begründung folgt!)} \end{aligned}$$

[Im Fall $I^+(h,i) = \emptyset$ ist diese Abschätzung trivial, denn leere Summen sind gleich Null. Aus $I^+(h,i) \neq \emptyset$ folgt $\sum_{k\in I^+(h,i)} p_{hk}^{(L)} \leq 1$ und $\sum_{k\in I^+(h,i)} p_{ik}^{(L)} \geq |I^+(h,i)| \cdot \delta \geq \delta$.]

Induktiv folgt für alle $\nu \geq 1$ zusammen mit $m_j^{(L)} \geq \delta$:

$$M_j^{(\nu L)} - m_j^{(\nu L)} \leq (1-\delta)^{\nu-1} (M_j^{(L)} - m_j^{(L)}) \leq (1-\delta)^{\nu}.$$

Wegen der Monotonie der Folgen $(m_j^{(n)})_{n\geq 1}$ und $(M_j^{(n)})_{n\geq 1}$ folgt damit die Konvergenz beider Folgen gegen ein und denselben Grenzwert, den wir mit ρ_j bezeichnen. Wir zeigen, dass diese Konvergenz exponentiell schnell verläuft: dazu sei $n = \lambda L + \nu$ mit $0 \leq \nu < L$. Damit gilt unter Verwendung von $M_j^{(n+L)} - m_j^{(n+L)} \leq (1-\delta)(M_j^{(n)} - m_j^{(n)})$:

$$\begin{aligned} M_j^{(n)} - m_j^{(n)} &\leq (1-\delta)^{\lambda}(M_j^{(\nu)} - m_j^{(\nu)}) \\ &\leq (1-\delta)^{n/L} \cdot (1-\delta)^{-\nu/L}(M_j^{(\nu)} - m_j^{(\nu)}) \\ &\leq \eta^n (1-\delta)^{-1}, \end{aligned}$$

wobei wir $\eta := (1-\delta)^{1/L}$ gesetzt und $(1-\delta)^{-\nu/L} \leq (1-\delta)^{-1}$ sowie $M_j^{(\nu)} - m_j^{(\nu)} \leq 1$ ausgenutzt haben. Mit $\eta < 1$ folgt die exponentiell schnelle Konvergenz sowohl der Maxima- und Minima- als auch damit aller Koeffizientenfolgen in den Potenzen von P gegen die jeweiligen Grenzwerte.

Durch Grenzübergang $n \to \infty$ folgt aus

$$p_{ik}^{(n+1)} = \sum_j p_{ij}^{(n)} p_{jk}$$

die Gleichung $\rho = \rho P$. Als Grenzwert von endlichdimensionalen Wahrscheinlichkeitsvektoren muss auch ρ ein Wahrscheinlichkeitsvektor. Diese Wahrscheinlichkeitsverteilung ρ ist eindeutig bestimmt: für jede Wahrscheinlichkeitsverteilung ρ' mit $\rho' = \rho' P$ folgt $\rho' = \rho' P^n$. Durch Grenzübergang folgt $\rho' = \rho' P^\infty$, woraus sich wegen $\rho'_k = \sum_j \rho'_j p_{jk}^{(\infty)} = \sum_j \rho'_j \rho_k = \rho_k$ sofort $\rho = \rho'$ ergibt. □

Beispiel. Die Matrix

$$P := \frac{1}{4} \cdot \begin{pmatrix} 0 & 3 & 1 \\ 2 & 0 & 2 \\ 1 & 0 & 0 \end{pmatrix}$$

ist zeilenstochastisch, also die Übergangsmatrix eines homogenen Markov-Prozesses. P enthält 4 Nullen, aber die Matrix P^4 enthält nur positive Einträge:

$$P^4 = \frac{1}{64} \cdot \begin{pmatrix} 25 & 18 & 21 \\ 32 & 15 & 17 \\ 24 & 30 & 10 \end{pmatrix}.$$

Damit sind die Voraussetzungen zur Anwendung des obigen Satzes erfüllt. Es existiert also eine Grenzverteilung $\rho = (\rho_1, \rho_2, \rho_3)$ mit $\rho P = \rho$. Insgesamt erhält man das Gleichungssystem

$$\begin{aligned} \rho_1 + \rho_2 + \rho_3 &= 1 \\ \tfrac{1}{2}\rho_2 + \rho_3 &= \rho_1 \\ \tfrac{3}{4}\rho_1 &= \rho_2 \\ \tfrac{1}{4}\rho_1 + \tfrac{1}{2}\rho_2 &= \rho_3, \end{aligned}$$

woraus sich nach leichter Rechnung

$$\rho_1 = \frac{8}{19}, \quad \rho_2 = \frac{6}{19} \quad \text{und} \quad \rho_3 = \frac{5}{19}$$

ergeben. Das heißt: Egal in welchem der drei Zustände man startet, man ist nach 100 Schritten ungefähr mit Wahrscheinlichkeit 8/19 im Zustand 1, mit Wahrscheinlichkeit 6/19 im Zustand 2 und mit Wahrscheinlichkeit 5/19 im Zustand 3.

□

2.12 Bemerkung

- ρ_j gibt an, wie oft Zustand j bei unendlicher Laufzeit im Mittel besucht wird.
- Die Bedingung $(\exists L : \forall i, j : p_{ij}^{(L)} > 0)$ fordert, dass es eine bestimmte Schrittlänge L gibt, so daß man in L Schritten aus jedem Zustand mit Wahrscheinlichkeit > 0 in jeden anderen gelangen kann. Man überlegt sich, dass dann auch $\forall i, j : p_{ij}^{(L+1)} > 0$ und somit induktiv $\forall K > L : p_{ij}^{(K)} > 0$ gilt.
- Aufgrund der exponentiell schnellen Konvergenz erhält man schnell gute Abschätzungen der ρ_j.
- Die Grenzverteilung ρ kann als Ranking der Zustände aus I interpretiert werden.

◇

Die Beziehung zu unserem PageRank-Konzept stellt folgendes Resultat her, bei dem wir aus technischen Gründen annehmen wollen, dass in unserem Webgraphen $G_g = (V_g, E_g)$ alle Knoten mindestens Ausgrad 1 und Eingrad 1 haben:

2.13 Korollar Die PageRank-Funktion $r : V_g \to \mathbb{R}$ aus (2.1),

$$r(v) = \frac{p}{|V_g|} + (1-p) \sum_{v' \in i(v)} \frac{r(v')}{o_{v'}}$$

ist unter der Nebenbedingung $\sum_{v \in V_g} r(v) = 1$ und $\forall v \in V_g : r(v) \geq 0$ wohldefiniert. Die Iteration

$$r^k(v) = \frac{p}{|V_g|} + (1-p) \sum_{v' \in i(v)} \frac{r^{k-1}(v')}{o_{v'}}$$

konvergiert für eine beliebige Startverteilung $r^0 : V_g \to [0,1]$, $\sum_{v \in V_g} r^0(v) = 1$, gegen r. ◇

Beweis: Wir setzen $N := |V_g|$ und definieren eine gewichtete Adjazenzmatrix $A \in \mathbb{R}^{N \times N}$ durch

$$A_{vv'} := \begin{cases} \frac{1}{o_{v'}} & \text{falls } (v', v) \in E_g, \\ 1 & \text{falls } v = v' \text{ und } o(v) = \emptyset, \\ 0 & \text{sonst.} \end{cases}$$

Hierbei liege eine beliebige aber feste Reihenfolge der Knoten des Graphen $G_g = (V_g, E_g)$ zu Grunde. Dann kann die PageRank-Gleichung (2.1) in Matrixnotation geschrieben werden als

$$r = \frac{p}{N} \cdot 1_N + (1-p)Ar, \tag{2.4}$$

wobei $r \in \mathbb{R}^N$, $1_N := (1, \ldots, 1)^\top$, $r := (r(v))_{v \in V_g}$ (in der zu den Matrixeinträgen passenden Reihenfolge). Wegen der Nebenbedingung gilt

$$1_N^\top r = 1.$$

Erweitern wir (2.4) mit letzterer Beziehung, erhalten wir

$$\begin{aligned} r &= \frac{p}{N} 1_N 1_N^\top r + (1-p)Ar \\ &= \left(\frac{p}{N} J_N + (1-p)A\right) r, \end{aligned}$$

wobei J_N diejenige $N \times N$-Matrix bezeichnet, deren Einträge alle gleich 1 sind. Laut Konstruktion gilt $\forall v' \in V_g : \sum_{v \in V_g} A_{vv'} = 1$ sowie

$$\forall 1 \leq j \leq N : \frac{1}{N} \sum_{i=1}^{N} (J_N)_{ij} = 1.$$

Daher und wegen $0 \leq p \leq 1$ sind auch die Spaltensummen der Matrix

$$M := \frac{p}{N} J_N + (1-p)A$$

gleich 1, d.h. M ist bzgl. der Spalten stochastisch. Da weiterhin $\forall i, j : M_{ij} > 0$, gibt es nach dem vorhergehenden Satz über Markov-Ketten (in einer analogen Version für Matrizen, die bezüglich ihrer Spalten stochastisch sind) eine eindeutige Lösung $Mr = r$ mit $r \geq 0$ und $\sum_i r_i = 1$. Sei $r^0 : V_g \to [0, 1]$ eine Startverteilung, dann gilt

$$r^k = Mr^{k-1},$$

also nach k-facher Iteration

$$r^k = M^k r^0.$$

Die Konvergenz der Iteration folgt dann aus der Konvergenz von $k \mapsto M^k$ für $k \to \infty$. □

Das dritte — von der Anfrage unabhängige (!) — Rankinggewicht der Webseite $u \in V_g$ wird dann als

$$w_u^3 := r(u)$$

gewählt.

Ranking: Zusammenfassung

Ein Gesamtranking für eine Webseite $u \in V_g$ zu einer Anfrage q wird nun aus den Einzelrankings $w_{u,q}^1, w_{u,q}^2$ und w_u^3 gebildet, etwa durch Linearkombination. Genaueres hierzu, z.B. die in Suchmaschinen tatsächlich gewählten Gewichtungen der Einzelrankings sind i.a. Betriebsgeheimnis der Betreiber der Suchmaschinen.

2.14 Bemerkung Die Variante *personalized PageRank* ersetzt die Gleichverteilung $\frac{1}{N}1_N$ durch einen personenspezifischen Verteilungsvektor $\nu \in \mathbb{R}^N$:

$$r_\nu(v) = p\nu_v + (1-p) \sum_{v' \in i(v)} \frac{r_\nu(v')}{|o(v')|}$$

ν soll das Surfverhalten einer bestimmten Person modellieren. Man erhält einen personalisierten PageRank $w_u^{3,\nu} := r_\nu(u)$. ◇

2.3.2 Strukturierung in Hubs und Authorities

Oft sind Anfragen nur vage oder unspezifisch. Gesucht sind dann oft Überblicksseiten zu einem bestimmten Thema bzw. geeignete Einstiegspunkte für eine weitere Suche. Seiten, die viele (gute) Links zu kompetenten Seiten bzgl. eines Themas enthalten, nennen wir im folgenden *Hubs*. Eine kompetente Seite zu einem Thema nennen wir *Authority* (bzgl. dieses Themas).
Der im folgenden vorgestellte Algorithmus ermittelt zu einer gegebenen Anfrage geordnete Mengen von Hubs und Authorities inclusive Ranggewichten.

Grobvorgehen:

- Ähnlich wie beim PageRank:
  - Aufbau eines Web-(Teil-) Graphen
  - Iterative Berechnung von Rankingwerten
- Anders als bei PageRank:
  - Graph und Ranking sind anfrageabhängig
  - Neuberechnung des Rankings bei jeder Anfrage nötig

Gegeben sei eine übliche Termanfrage σ (z.B. $\sigma \in L^+$) an eine Suchmaschine. Das Ergebnis sei eine Menge $R_\sigma \subseteq V$ von Webseiten. Eventuell kann man R_σ noch weiter ausdünnen, indem man nur die besten n Webseiten aus R_σ bzgl. eines ad hoc-Kriteriums verwendet. Wir bezeichnen die Menge R_σ als *Root-Set* zu σ.

Aufbau des Graphen zur Anfrage σ

Die zu Grunde liegende Idee ist, dass sich zur Anfrage σ relevante Seiten sowohl in R_σ als auch in der Nachbarschaft (bzgl. der Linkstruktur) von R_σ befinden. Wir bestimmen die Menge

$$\begin{aligned} V[R_\sigma] &:= \{v \in V \setminus R_\sigma | \exists v' \in R_\sigma : (v', v) \in E \vee (v, v') \in E\} \\ &= \left[\bigcup_{v \in R_\sigma} (i(v) \cup o(v))\right] \setminus R_\sigma \end{aligned}$$

aller Vorgänger- und Nachfolgerknoten zu Knoten aus R_σ, die selber nicht in R_σ liegen.
In der Praxis ist es wichtig, hierbei zu einem $v \in R_\sigma$ nicht alle Seiten $i(v)$ zu verwenden, da i_v oft sehr groß ist. Man beschränkt sich in diesem Fall auf Teilmengen $\tilde{i}(v) \subseteq i(v)$ einer vorgewählten Maximalgröße $|\tilde{i}(v)| \leq m$. Anstatt $V[R_\sigma]$ erhält man eine kleinere Knotenmenge $\tilde{V}[R_\sigma] \subseteq V[R_\sigma]$.
Aus $\tilde{V}[R_\sigma]$ bestimmt man das sogenannte *Base-Set* B_σ via

$$(B_\sigma, \tilde{E}_\sigma) := G|_{\tilde{V}[R_\sigma] \cup R_\sigma},$$

siehe Abb. 2.5. Für die im folgenden dargestellten Techniken spielt es keine Rolle, ob man von der Knotenmenge $\tilde{V}[R_\sigma]$ oder $V[R_\sigma]$ ausgeht. Die im folgenden vorgestellten Konzepte gelten für beide Fälle gleichermaßen und werden anhand der Knotenmenge $V[R_\sigma]$ entwickelt.
Es stellt sich heraus, dass es sinnvoll ist, Links innerhalb einer Domäne nicht zu betrachten, daher betrachtet man schließlich den Graphen $G_\sigma := (B_\sigma, E_\sigma)$ mit

$$E_\sigma = \{(v, v') \in \tilde{E}_\sigma \mid H(v) \neq H(v')\},$$

wobei H wie üblich eine Zuordnung von Webseiten zu Domänen bzgl. Hostrechnern ist.

Bestimmung von Hub- und Authority-Gewichten

Gesucht sind nun

- eine Authority-Gewichtung $a : B_\sigma \to [0, 1]$ (genauer $a = a_\sigma$) und
- eine Hub-Gewichtung $h : B_\sigma \to [0, 1]$ (genauer $h = h_\sigma$).

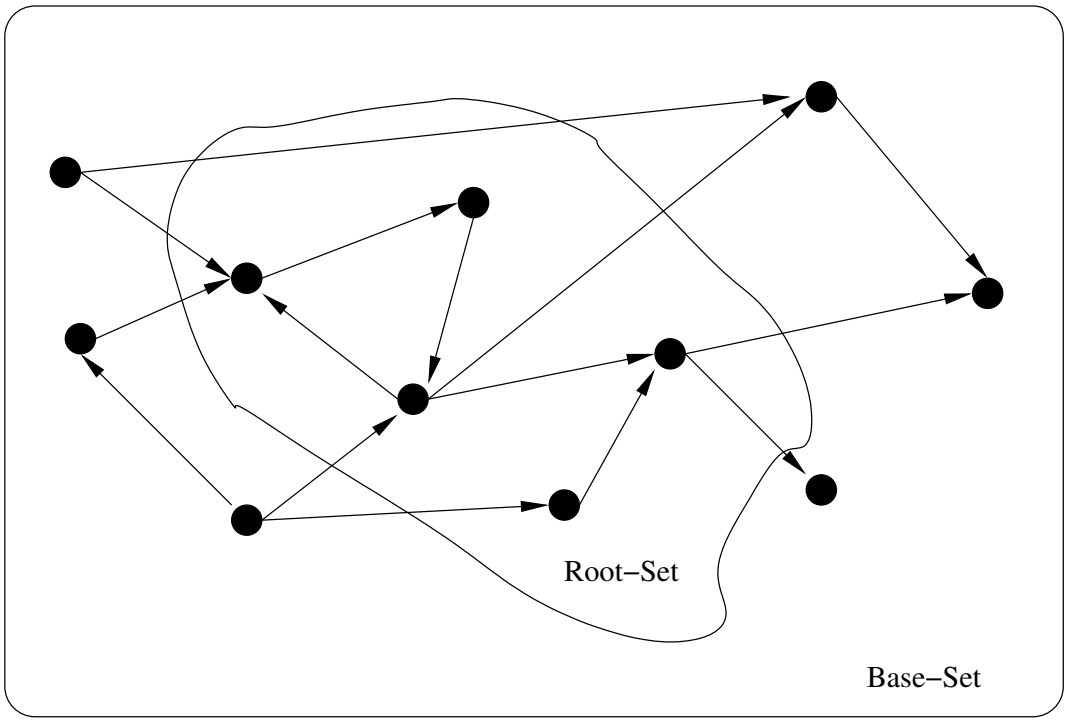


Abbildung 2.5: Root-Set R_σ und Base-Set B_σ.

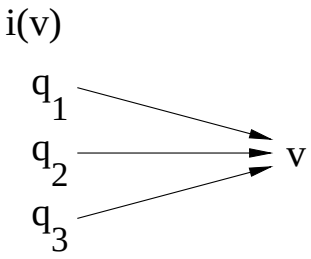
Für eine beliebige aber feste Ordnung der Knoten in B_σ und $N := |B_\sigma|$ fassen wir die Gewichtungen im folgenden auch als Vektoren $a, h \in [0,1]^N$ auf.

Idee:

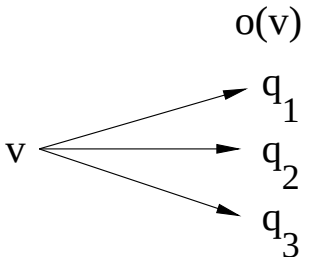
- $v \in B_\sigma$ erhält sein Authority-Gewicht gemäß den Hub-Gewichten seiner Vorgänger $i(v)$ und

- $v \in B_\sigma$ erhält sein Hub-Gewicht gemäß den Authority-Gewichten seiner Nachfolger $o(v)$.

Überblick:



v als *Authority* $\qquad a(v) = \sum_{u \in i(v)} h(u) \quad (1)$



v als *Hub* $\qquad h(v) = \sum_{u \in o(v)} a(u) \quad (2)$

Bezeichnet $A := A[G_\sigma]$ die Adjazenzmatrix von G_σ bezüglich der gewählten Knotenreihenfolge, so lassen sich (1) und (2) schreiben als

$$a = A^\top h \text{ und } h = Aa.$$

Das Auffinden einer sinnvollen Hub- und Authority-Gewichtung beruht nun darauf, a als Eigenvektor der Matrix $A^\top A$ und h als Eigenvektor der Matrix $AA^\top$ zu wählen. Dann gelten

$$A^\top Aa = \lambda a \text{ und } AA^\top h = \mu h$$

für geeignete Eigenwerte λ und μ, so dass die Gewichtungen unter Iterationen der Vorschriften (1) und (2) bis auf Skalierungen invariant sind.
Wie im Falle von PageRank finden wir geeignete Eigenvektoren per Fixpunktiteration:

2.15 Satz Seien $a_0, h_0 \in \mathbb{R}^N$ geeignet gewählte Startwerte und A wie oben definiert, dann konvergieren die Iterationen

$$a_k := \frac{A^\top h_{k-1}}{\|A^\top h_{k-1}\|_\infty}$$

und

$$h_k := \frac{Aa_k}{\|Aa_k\|_\infty}$$

für $k \to \infty$ gegen Grenzwerte $a^* := \lim_{k\to\infty} a_k$ und $h^* := \lim_{k\to\infty} h_k$. ◇

Was in diesem Zusammenhang eine geeignete Wahl der Startwerte ist, ergibt sich im Verlauf des Beweises.

Beweis: Wir betrachten hier nur die Iteration bzgl. a_k. Der Beweis für die andere Iteration verläuft analog. Einmaliges Einsetzen ergibt

$$a_k = \frac{A^\top Aa_{k-1}}{\|A^\top Aa_{k-1}\|_\infty}.$$

$A^\top A \in \mathbb{R}^{N\times N}$ ist symmetrisch. Ist $m :=\text{rang}(A^\top A)$, so gibt es eine orthonormale $m \times N$-Matrix U mit

$$A^\top A = U^\top \text{diag}(\lambda_1, \ldots, \lambda_m)U,$$

wobei $\lambda_i \in \mathbb{R}_{\neq 0}$ die von Null verschiedenen Eigenwerte von $A^\top A$ und die Zeilenvektoren $u_1, \ldots, u_m$ von U die zugehörigen Eigenvektoren sind.
Sei weiter $u_{m+1}, \ldots, u_N$ eine Basis von $\text{Kern}(A^\top A)$. Schreibt man den Startwert a_0 als

$$a_0 = \alpha_1 u_1 + \ldots + \alpha_m u_m + \underbrace{\alpha_{m+1}u_{m+1} + \ldots + \alpha_N u_N}_{=:r(a_0)},$$

dann gilt $A^\top A r(a_0) = 0$. Dann ist

$$a_1 = \frac{1}{\|A^\top A a_0\|_\infty}(\lambda_1\alpha_1 u_1 + \ldots + \lambda_m \alpha_m u_m).$$

Nun nehmen wir an, dass a_0 so gewählt ist, dass $\alpha_1 \neq 0$, d.h. $u_1^\top a_0 \neq 0$. (Es ist klar, dass solche a_0 existieren. Wie man in der Praxis a_0 wählen kann, besprechen wir später.)
Die Eigenwerte seien O.B.d.A. betragsmäßig absteigend sortiert,

$$|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_m|,$$

wobei wir weiterhin $|\lambda_1| > |\lambda_2|$ annehmen. Dann erhält man durch k-fache Iteration

$$a_k = \frac{1}{c_k}\lambda_1^k \left(\alpha_1 u_1 + \sum_{\ell=2}^{m} \left(\frac{\lambda_\ell}{\lambda_1}\right)^k \alpha_\ell u_\ell\right), \tag{2.5}$$

wobei $c_k = \prod_{\ell=0}^{k-1} \|A^\top A a_\ell\|$ den akkumulierten Normalisierungsfaktor bezeichnet. Da für $2 \leq \ell \leq m$

$$\lim_{k\to\infty} \left(\frac{\lambda_\ell}{\lambda_1}\right)^k \to 0$$

gilt, folgt $\lim_{k\to\infty} a_k = u_1/\|u_1\|_\infty$.
Setzt man genauer

$$a'_{k+1} := A^\top A a_k$$

und

$$a_{k+1} := \frac{a'_{k+1}}{\|a'_{k+1}\|_\infty},$$

dann folgt weiterhin $\lim_{k\to\infty}\|a'_k\|_\infty = |\lambda_1|$.
Gibt es mehrere betragsmäßig größte Eigenwerte, unterscheiden wir zwei Fälle. Gilt

$$\lambda_1 = \ldots = \lambda_r \text{ und } |\lambda_r| > |\lambda_{r+1}| \geq \ldots \geq |\lambda_m|,$$

so erhalten wir anstatt (2.5)

$$a_k = \frac{1}{c_k}\lambda_1^k \left(\sum_{\ell=1}^{r} \alpha_\ell u_\ell + \sum_{\ell=r+1}^{m} \left(\frac{\lambda_\ell}{\lambda_1}\right)^k \alpha_\ell u_\ell \right),$$

so dass die Iteration gegen einen Vektor aus $\text{span}(u_1, \ldots, u_r)$ konvergiert. Dieser Vektor ist abhängig vom Startwert a_0.
Gilt nur

$$|\lambda_1| = \ldots = |\lambda_r| \text{ und } |\lambda_r| > |\lambda_{r+1}| \geq \ldots \geq |\lambda_m|,$$

so divergiert die Interation i.a. gegen unterschiedliche Grenzvektoren. Aus diesen Grenzvektoren können jedoch wiederum durch Linearkombination die Eigenwerte der λ_i bestimmt werden, siehe [9], Kapitel 15.4.

□

Der Grenzwert a^* (bzw. h^*) entspricht somit einer normierten Version eines Eigenvektors zum dominanten Eigenwert λ_1 von $A^\top A$ (bzw. μ_1 von $AA^\top$). Ist der dominante Eigenwert nicht einfach, liegt a^* (bzw. h^*) im Eigenraum des Eigenwerts. Die Konvergenzgeschwindigkeit (im ersten Fall) hängt vom Quotienten λ_2/λ_1 ab.
Zu klären ist noch, wie man geeignete Startwerte für a_0 und h_0 wählt. Hierzu benötigen wir die zwei folgenden Resultate.

2.16 Lemma Ist $M \in \mathbb{R}^{N\times N}$, so dass $i \sim_M j :\Leftrightarrow M_{ij} \neq 0$ Äquivalenzrelation auf $[1:N]$ ist, dann existieren eine Permutationsmatrix $P \in \{0,1\}^{N\times N}$ und ein $k \in \mathbb{N}$, so dass

$$P^\top M P = \text{diag}(D_1, \ldots, D_k)$$

wobei $D_i \in \mathbb{R}_{\neq 0}^{d_i \times d_i}$ und $\sum_{i=1}^{k} d_i = N$. ◇

Beweis: Übung! □

2.17 Satz (Perron-Frobenius) Ist $M \in \mathbb{R}_{\geq 0}^{N\times N}$ nicht-negative Matrix und existiert ein $k \in \mathbb{N}$, so da $M^k \in \mathbb{R}_{>0}^{N\times N}$, dann besitzt M einen reellen Eigenwert $r > 0$, so dass

1. für alle Eigenwerte $\lambda \neq r$ gilt $r > |\lambda|$ und
2. ein strikt positiver Eigenvektor $v \in \mathbb{R}_{>0}^{N}$ zu r existiert.

◇

Beweis: Ohne Beweis. (Für einen elementaren Beweis, siehe [10].) □

Die Matrix $A^\top A$ ist symmetrisch. Weiterhin gehen wir davon aus, dass A keine Nullspalten enthält (entsprechend Knoten ohne eingehende Kanten). Existieren solche Spalten, bilden wir einfach die Streichungsmatrix der Matrix A bzgl. dieser Spalten. Die entsprechenden Knoten bzw. Webseiten erhalten dann den Rankingwert 0. Dann sieht man leicht (Übung!), dass es $e \in \mathbb{N}$ gibt, so dass $M := (A^\top A)^e$

die Voraussetzungen von Lemma 2.16 erfüllt, d.h. es existiert eine Permutationsmatrix P, so dass

$$P^\top MP = \mathrm{diag}(D_1, \ldots, D_k).$$

Also kann man die Knoten des Graphen O.B.d.A. so anordnen, dass M von der Form $\mathrm{diag}(D_1, \ldots, D_k)$ ist, wobei alle D_i strikt positiv sind. Dann ist für $w \in \mathbb{R}^N$

$$P^\top MPw = ((D_1 w_1)^\top, \ldots, (D_k w_k)^\top)^\top,$$

also

$$M^\ell w = ((D_1^\ell w_1)^\top, \ldots, (D_k^\ell w_k)^\top)^\top, \tag{2.6}$$

wobei w_i der i-te Block von w (der Länge d_i) ist. Wegen Satz 2.17 hat jede Matrix D_i einen größten Eigenwert λ_i mit strikt positivem Eigenvektor v_i. Wählt man nun 1_{d_i} als Startwert, dann ist $\langle v_i, 1_{d_i}\rangle \neq 0$ und wegen der Eigenschaften von λ_i konvergiert die Folge $\ell \mapsto D_i^\ell 1_{d_i}/\|D_i^\ell 1_{d_i}\|_\infty$ wie im Beweis zu Satz 2.15 gesehen gegen einen Vektor v_i^*. Wegen 2.6 konvergiert dann auch die Gesamtiteration für den Startvektor $1_N = (1_{d_1}^\top, \ldots, 1_{d_k}^\top)^\top$.
Kleinberg [11] berichtet, dass in Tests ca. $k = 20$ Iterationsschritte für eine gute Approximation a_k, h_k der Authority- und Hub-Gewichte ausreichen. Es sei kritisch angemerkt, dass in anderen Quellen davor gewarnt wird, dass nicht selten $\lambda_2/\lambda_1 \sim 1$, d.h. schlechte Konvergenzeigenschaften vorliegen.
Es gibt noch weitere, im Sinne obiger Iteration konsistente Wahlen für Hub- und Authority-Gewichte. Grundlage ist folgendes Resultat, das die Eigenwerte und -vektoren von $A^\top A$ und $AA^\top$ in Beziehung setzt:

2.18 Satz $A^\top A$ und $AA^\top$ besitzen dieselben Eigenwerte mit denselben Vielfachheiten. Weiterhin kann man eine Matrix V aus Eigenvektoren zu $A^\top A$ so wählen, dass AV aus einem vollständigen System von Eigenvektoren von $AA^\top$ besteht. $\diamond$

Beweis: Sei v Eigenvektor von $A^\top A$ zum Eigenwert λ, also $A^\top Av = \lambda v$. Multiplikation mit A liefert

$$AA^\top Av = \lambda Av,$$

d.h., Av ist Eigenvektor von $AA^\top$ zum Eigenwert λ.
Ist $V \in \mathbb{R}^{N\times m}$ Matrix aus linear unabhängigen m Eigenvektoren von $A^\top A$ zum Eigenwert λ, die eine Basis des Eigenraums zu λ bilden, so gilt

$$AA^\top AV = \lambda AV \text{ mit } AV \in \mathbb{R}^{N\times m}.$$

Da $A^\top AV = \lambda V$, muss $\mathrm{rang}(AV) = \mathrm{rang}(V)$ gelten, also ist λ auch m-facher Eigenwert von $AA^\top$. Eine analoge Argumentation für die restlichen Eigenwerte von $A^\top A$ sowie eine Umkehrüberlegung für Eigenwerte von $AA^\top$ zeigen die restliche Behauptung. $\square$

2.19 Korollar Ist $v \in \mathbb{R}^N$ normierter Eigenvektor von $A^\top A$ zum positiven Eigenwert λ, so gilt mit den Startwerten $a_0 := v$ und $h_0 := Av$ für alle $k \geq 1$:

$$a_k = v \quad \text{sowie} \quad h_k = \frac{Av}{\|Av\|_\infty}.$$

◇

Beweis: Wegen $A^\top Av = \lambda v, \lambda > 0, \|v\|_\infty = 1$ und $a_k = A^\top A a_{k-1}/\|A^\top A a_{k-1}\|_\infty$ folgt

$$a_1 = \frac{A^\top Av}{\|A^\top Av\|_\infty} = \frac{\lambda v}{\|\lambda v\|_\infty} = \frac{\lambda}{|\lambda|\|v\|_\infty} v = v.$$

Induktion liefert $a_1 = v, \forall k \geq 1$. Wegen $h_k = Aa_k/\|Aa_k\|_\infty$ und $a_k = v$ folgt $h_k = Av/\|Av\|_\infty, \forall k \geq 1$. □

2.20 Beispiel (Nach Kleinberg [11]) Die Anfrage `jaguar*` zur Suche nach der Singular- oder Pluralform des Wortes Jaguar liefert für die drei vom Betrag größten Eigenwerte folgende Authorities:

Prinzipaler Eigenwert:

Gewicht	URL	Titel
0.37	http://www2.ecst.csuchico.edu/∼jschlich/Jaguar/jaguar.html	
0.347	http://www-und.ida.liu.se/∼t94patsa/jserver.html	
0.292	http://tangram.informatik.uni-kl.de:8001/∼rgehm/jaguar.html	
0.287	http://www.mcc.ac.uk/dlms/Consoles/jaguar.html	Jaguar Homepage

Dies sind Webseiten zur Spielekonsole `Jaguar` der Firma Atari.

Zweitgrößter Eigenwert:

Gewicht	URL	Titel
0.255	http://www.jaguarsnfl.com/	Official Jacksonville Jaguars NFL Website
0.137	http://www.nando.net/SportServer/football/nfl/jax.html	Jacksonville Jaguars Homepage
0.133	http://www.ao.net/∼brett/jaguar/index.html	Brett's Jaguar Page
0.11	http://www.usatoday.com/sports/football/sfn/sfn30.htm	Jacksonville Jaguars

Die hier gefundenen Authorities sind Webseiten, die sich mit der Football-Mannschaft `Jacksonville Jaguars` beschäftigen.

Drittgrößter Eigenwert:

Gewicht	URL	Titel
0.227	http://www.jaguarvehicles.com/	Jaguar Cars Global Home Page
0.227	http://www.collection.co.uk/	The Jaguar Collection - Official Web site
0.211	http://www.moran.com/sterling/sterling.html	
0.211	http://www.coys.co.uk/	

Bei diesen Webseiten handelt es sich um Seiten zum Autohersteller bzw. zur Fahrzeugmarke `Jaguar`.

◇

2.3.3 Verfeinerte Hub- und Authority-Gewichtung

Folgende Probleme stellen sich bei Verwendung des Roh-Algorithmus aus dem letzten Abschnitt heraus:

1. Links sind nicht geeignet gewichtet:
   - Zeigen viele Seiten einer Domäne auf eine bestimmte Seite, so wird diese Seite ein hohes Authority-Gewicht erhalten. Wie bei PageRank erwähnt, kann dies zu Manipulationen und unangemessen hohen Bewertungen führen.
   - Zeigt eine einzelne Seite auf viele Seiten derselben Domäne, so läuft sie Gefahr, ein zu hohes Hub-Gewicht zugewiesen zu bekommen.
2. Automatisch generierte Links (Werbung, Link auf Webmaster oder Provider, etc.) führen aufgrund ihrer Masse zu falschen Authorities.
3. Drift zu einem anderen (aber eventuell verwandten) Thema: Liegen in B_σ bzw. schon in R_σ Seiten zu einem anderen Thema, die stärker verlinkt sind, als die Seiten des gesuchten Themas, erhalten diese Seiten eventuell Vorrang gegenüber den gesuchten Seiten (*Topic drift*).

Zu 1.: Maßnahmen zur Gewichtung der Links:

- Zu einer Webseite $v \in V$ kann eine Domäne $\mu \in \mathcal{H}$ nur eine Stimme abgeben. Jeder Link (v', v) mit $v' \in i_\mu(v)$ erhält somit Gewicht $1/|i_\mu(v)|$.
- Eine Webseite $v \in V$ kann nur eine Stimme für eine Domäne $\mu \in \mathcal{H}$ abgeben. Jeder Link (v, v') mit $v' \in o_\mu(v)$ erhält somit Gewicht $1/|o_\mu(v)|$.

Zusammenfassend realisiert man (für eine geeignete Nummerierung der Knoten) Kantengewichtungen

$$\alpha, \eta : E_\sigma \to [0, 1]$$

für Authorities und Hubs via

$$\alpha : (v, v') \mapsto \begin{cases} 0 & \text{falls } H(v) = H(v') \text{ oder } i_{H(v)}(v') = \emptyset, \\ \frac{1}{|i_{H(v)}(v')|} & \text{sonst,} \end{cases}$$

sowie

$$\eta : (v, v') \mapsto \begin{cases} 0 & \text{falls } H(v) = H(v') \text{ oder } o_{H(v')}(v) = \emptyset, \\ \frac{1}{|o_{H(v')}(v)|} & \text{sonst.} \end{cases}$$

Analog zum letzten Abschnitt versuchen wir nun wieder, konsistente Knotengewichte $a \in \mathbb{R}^N$ für ein Authority-Ranking und $h \in \mathbb{R}^N$ für ein Hub-Ranking zu bestimmen. Die Iterationsvorschriften sind hier

$$a_{k+1}(v) := \sum_{(v',v)\in E_\sigma} h_k(v')\alpha(v',v) \tag{2.7}$$

und

$$h_k(v) := \sum_{(v,v')\in E_\sigma} a_k(v')\eta(v,v'). \tag{2.8}$$

2.21 Satz Die Iteration (2.7), (2.8) konvergiert mit Startwerten $a_0 = 1_N$ und $h_0 = 1_N$ gegen Grenzwerte $a^* = \lim_{k\to\infty} a_k$ und $h^* = \lim_{k\to\infty} h_k$. ◇

Beweis: Schreibe (2.7) und (2.8) mit

$$H_{vv'} := \eta(v',v)$$

und

$$A_{vv'} := \alpha(v,v')$$

(man bemerke die Spiegelung der Komponenten) als Matrixoperationen $a_{k+1} = Ah_k$ und $h_k = Ha_k$. Dann gilt für Knoten v, v'

$$A_{vv'} \neq 0 \Leftrightarrow H_{v'v} \neq 0$$

also, weil A, H nur Einträge ≥ 0 besitzen

$$(HA)_{v'v} \neq 0 \Leftrightarrow (HA)_{vv'} \neq 0.$$

Weiterhin gilt für einen Knoten v

$$(HA)_{vv} = 0 \Leftrightarrow \forall v' : H_{vv'} = 0 \wedge A_{v'v} = 0,$$

d.h. eine 0 in der Diagonalen von HA impliziert eine Nullzeile in H und Nullspalte in A. Solche Zeilen und Spalten lassen wir, wie im vorherigen Abschnitt beschrieben, durch Einschränkung auf die entsprechenden Streichungsmatrizen von H und A weg.
Wie im letzten Abschnitt gesehen, können wir nach dieser Vorbereitung durch Umnummerierung der Knoten erreichen, dass für $e \in \mathbb{N}$

$$(HA)^e = \mathrm{diag}(D_1, \ldots, D_k).$$

Im Gegensatz zur Rohversion des Hub- und Authority-Algorithmus sind die Matrizen D_i allerdings hier nicht mehr symmetrisch, so dass wir die Konvergenz der Iteration nicht mehr wie vorher beweisen können.
Dass die Iteration $m \mapsto (HA)^m 1_N / \|(HA)^m 1_N\|_\infty$ für $m \to \infty$ trotzdem konvergiert, zeigt folgendes Zusatzresultat:

2.22 Satz (ohne Beweis, siehe[10]) Ist $M \in \mathbb{R}_{\geq 0}^{N \times N}$ positive Matrix und existiert $K \in \mathbb{N}$, so das $M^K \in \mathbb{R}_{>0}^{N \times N}$, so gilt

$$M^m = r^m w v^\top + O(m^s |\lambda_2|^m),$$

wobei r der größte Eigenwert von M laut Satz 2.17 (Perron-Frobenius) und $w, v^\top$ Rechts- und Linkseigenvektoren von M bzgl. r ($Mw = rw$ und $v^\top M = rv^\top$) sind, die so normiert sind, dass $v^\top w = 1$. Weiter ist $s = m_2 - 1$, wobei m_2 die Vielfachheit des zweitgrößten Eigenwerts λ_2 von M ist. $\diamond$

Aus diesem Satz folgt $\lim_{m\to\infty}(M^m/r^m - wv^\top) = 0$, also konvergiert

$$m \mapsto (HA)^m 1_N / \|(HA)^m 1_N\|_\infty$$

für $m \to \infty$ gegen den Vektor $wv^\top 1_N / \|wv^\top 1_N\|_\infty$. $\square$

Zu 2. und 3.: Wir verwenden hier die Heuristik, dass Webseiten im Root-Set eher mit dem Anfragethema übereinstimmen, als solche im Base-Set. (Beachte, dass dies kein Widerspruch dazu sein muss, dass es im Base-Set bzgl. der Anfrage relevantere Seiten geben kann!)

Ansatz: Eliminiere Seiten aus B_σ, die unähnlich zu den Seiten aus dem Root-Set sind.

Durchführung: Sei $D_\sigma := \sqcup_{v \in R_\sigma} D(v)|_K$ ein Dokument bestehend aus der Konkatenation $\sqcup$ der jeweils ersten K Terme aller Dokumente $D(v)$ zu Seiten im Root-Set R_σ (in beliebiger Reihenfolge).

Sei $s : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ geeignetes Ähnlichkeitsmaß auf der Menge der Dokumente $\mathcal{D}$ (z.B. das Cosinusmaß aus §1). Wir bilden nun ein bereinigtes Base-Set

$$\tilde{B}_\sigma := \{v \in B_\sigma \mid s(D(v), D_\sigma) \geq \theta\}$$

und setzen

$$\tilde{G}_\sigma := G_\sigma|_{\tilde{B}_\sigma}.$$

Hierbei ist ein Schwellwert $\theta \geq 0$ geeignet zu wählen, z.B.

- Median aller $s(D(v), D_\sigma)$ für $v \in B_\sigma$,
- Median aller $s(D(v), D_\sigma)$ für $v \in R_\sigma$,
- prozentualer Anteil am Maximum: Für $0 \leq p \leq 1$ sei

$$\theta := p \cdot \max_{v \in B_\sigma} s(D(v), D_\sigma).$$

Alternativ kann man mit $w(v) := s(D(v), D_\sigma)$ einen Gewichtsvektor $w : B_\sigma \to [0, 1]$ bilden, den man via

$$a_{k+1}(v) = \sum_{(v,v') \in E_\sigma} h_k(v') w(v') \alpha(v, v')$$

(analog für (2.8)) in die Ranking-Berechnung einbaut. Für die Matrixoperation bedeutet dies eine Gewichtung der Zeilen bzw. Spalten. Die Gewichtung mit w verleiht bzgl. s zu D_σ unähnlichen Webseiten ein geringeres Gewicht als ähnlichen.

Kommentare zu 2.3.2 und 2.3.3

- Der Hub- und Authority-Ansatz führt zu wesentlich besseren Ergebnissen, als bloße Root-Set-Suche (Suchmaschinen-Suche).
- Ansatz ist qualitativ vergleichbar mit Google-Ansatz (PageRank mit Zuweisung von Linktext zu referenzierten Seiten).
- Die Erweiterung gemäß Abschnitt 2.3.3 bringt laut durchgeführten Tests deutliche Verbesserungen.
- Ein Problem sind zur Zeit noch die relativ langen Bearbeitungszeiten (30 Sekunden bis 1 Minute pro Anfrage), die laut Analysen jedoch alleine von den Zeiten zum Laden der Webseiten dominiert werden.

2.4 Suche nach verwandten Themen

Anfrage: Webseite $v \in V$

Gesucht: Webseiten aus V mit zu v „verwandtem“ Inhalt

Da „verwandter Inhalt“ nicht genau spezifiziert ist, versucht man wieder mit Hilfe geeigneter Heuristiken den Verwandheitsgrad von Webseiten zu beurteilen.

2.23 Beispiel Suche nach verwandten Webseiten ist z.B. möglich über die Suchmaschinen

- Netscape unter Verwendung des *What's related*-Dienstes (Alexa),
- Google bei der *erweiterten Suche*.

◇

Implementierungsdetails werden in beiden Fällen geheim gehalten. Wir besprechen hier zwei Algorithmen, die im Umfeld der Suchmaschine Google vorgeschlagen wurden.

2.4.1 Der Companion–Algorithmus

Der Companion-Algorithmus basiert auf dem verfeinerten Algorithmus zur Hub- und Authority-Gewichtung, vgl. 2.3.3. Wir gehen von einer angefragten Webseite $v \in V$ aus.

1. **Aufbau eines Web-Subgraphen zu** v

   Für eine beliebige Webseite $w \in V$ sei $(w_1, \ldots, w_{o_w}) \in V^{o_w}$ eine Folge bestehend aus allen per ausgehendem Link mit w verbundenen Webseiten. (Hierbei liege eine geeignete Ordnung zu Grunde, i.a. in der Reihenfolge in der die Links auf der Seite w auftreten.)

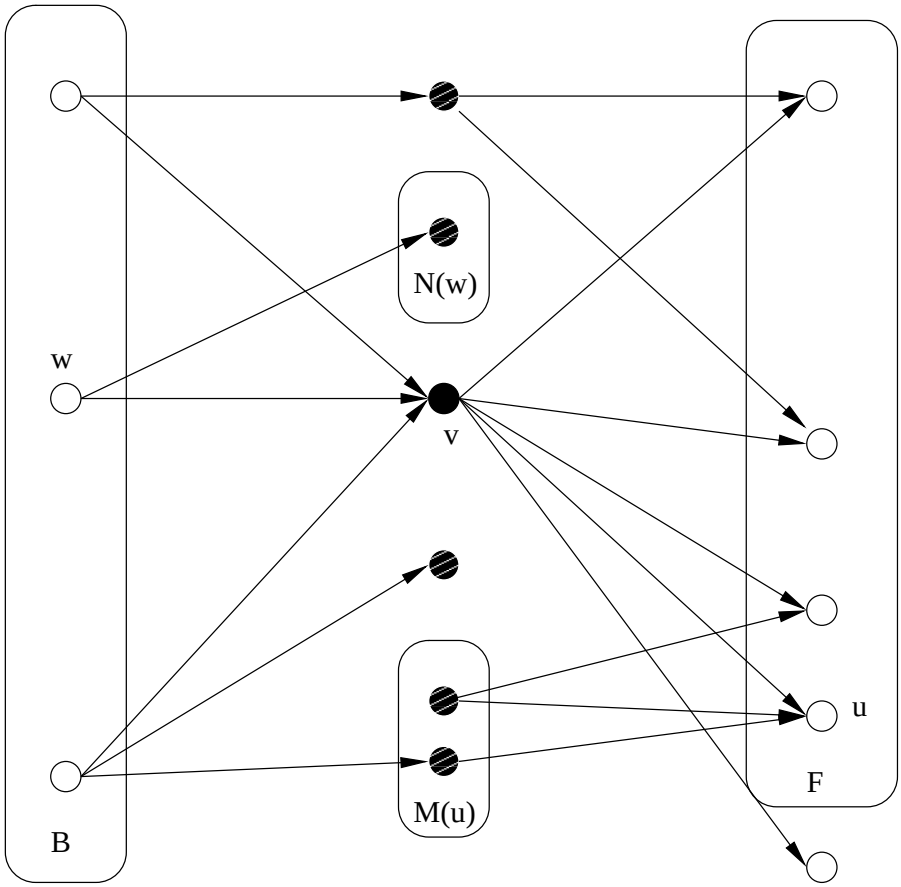


Abbildung 2.6: Idee zum Webgraphen des Companion-Algorithmus.

Die Idee zum Webgraphen des Companion-Algorithmus ist in Abb. 2.6 dargestellt: In Erweiterung des Hub-Authority-Graphen, werden nicht nur Seiten der Vorgänger- und Nachfolge-Ebene zu v betrachtet, sondern auch solche auf „derselben Ebene“.

Genauer wählt man nun Konstanten $b, f, b_f, f_b \in \mathbb{N}$ und bestimmt Mengen

- $B \subseteq_R i(v)$, so dass $|B| = \min\{i_v, b\}$, $\quad (B \sim \text{backward})$
- $F := \{v_\ell |\ 1 \leq \ell \leq \min\{o_v, f\}\}$, $\quad (F \sim \text{forward})$
- und für alle $u \in B$ mit $u_\ell = v$:

$$N(u) := \{u_{\ell - b_f}, \ldots, u_{\ell-1}, u_{\ell+1}, \ldots, u_{\ell+b_f}\}.$$

falls soviele u_i's existieren (die u_i's seien dabei notfalls so umnummeriert, dass Vorkommen von v ignoriert werden, also $v \notin N(u)$ gilt). Ansonsten setze $N(u) := o(u) - \{v\}$.

- Für ein $u \in F$ bezeichne $(u_1, \ldots, u_{i_u})$ eine Sortierung der Elemente aus $i(u) \setminus \{v\}$ mit
$$i_{u_1} \geq i_{u_2} \geq \ldots \geq i_{u_{i_u}}.$$
Setze dann
$$M(u) := \{u_\ell \mid 1 \leq \ell \leq \min\{i_u, f_b\}\}.$$

Mit diesen Konventionen bestimmt man schließlich den Web-Subgraphen

$$(V_v, E_v) := G_v := G|_{\{v\} \cup B \cup F \cup (\cup_{u \in B} N(u)) \cup (\cup_{u \in F} M(u))}.$$

2.24 Bemerkung In der Praxis arbeitet man oft mit einer *Stop-Liste* $S \subset V$ bestimmter Webseiten. Ist $v \notin S$, so werden alle Seiten aus S bei der Konstruktion von G_v ignoriert, ansonsten bleibt die Stop-Liste bei der Konstruktion von G_v unberücksichtigt. ◇

2.25 Beispiel In Versuchen wurden als Parameter $b \approx 2000$ und $b_f \approx 8$ gewählt. ◇

2. **Entfernen von Duplikaten**

Webseiten $u, u' \in V$ werden *Duplikate* genannt gdw.

(a) genügend ausgehende Links vorhanden sind,
$$o_u \geq 10 \wedge o_{u'} \geq 10,$$
und

(b) mehr als 95% der ausgehenden Links übereinstimmen,
$$\frac{|o(u) \cap o(u')|}{o_u} \geq 0.95 \wedge \frac{|o(u) \cap o(u')|}{o_{u'}} \geq 0.95.$$

Duplikate werden mittels des folgenden Algorithmus (in Pseudocode) zusammengefasst:

`while` $\exists u, u' \in V_v : u, u'$ Duplikate
  `add fresh node` u'' `to` V_v
  `add edges` $\{(i, u'') \mid i \in (i(u) \cup i(u')) - \{u, u'\}\}$ `to` E_v
  `add edges` $\{(u'', o) \mid o \in (o(u) \cup o(u')) - \{u, u'\}\}$ `to` E_v
  `delete` u, u' `from` V_v
  `delete` $\{(w, w') \in E_v \mid w \in \{u, u'\} \vee w' \in \{u, u'\}\}$ `from` E_v

```
end
```

3. **Berechnung von Hub- und Authority-Gewichten**

   Die Berechnung von Hub- und Authority-Gewichten wird analog zum verfeinerten Hub-Authority-Algorithmus 2.3.3 durchgeführt.

Schließlich stellen die Seiten aus V_v mit den höchsten Authority-Gewichten die zu v „verwandten“ Seiten dar.

2.4.2 Der Cocitation-Algorithmus

Der Cocitation-Algorithmus stellt eine — bezüglich der Komplexität der Rankingberechnung — kostengünstige Alternative zum Companion-Algorithmus dar.
Idee: Zu einer Anfrageseite $v \in V$ und einem Web-Subgraphen V_v bestimme für alle $u \in V_v$ die Anzahl von Webseiten die einen Link auf u *und* v besitzen.
Wie beim Companion-Algorithmus wählt man vorab geeignete Konstanten $b, b_f \in \mathbb{N}$ (mit gleicher Semantik wie beim Companion-Algorithmus).

1. **Aufbau eines Web-Subgraphen zu** v

   (a) Wähle $B \subseteq_R i(v)$, so dass $|B| = \min\{i_v, b\}$.

   (b) Für alle $u \in B$ mit $u_\ell = v$ (es sei wieder $(u_1, \ldots, u_{o_u})$ eine geeignete Nummerierung der von u verlinkten Webseiten) setze

   $$N(u) := \{u_{\ell-b_f}, \ldots, u_{\ell-1}, u_{\ell+1}, \ldots, u_{\ell+b_f}\}$$

   falls soviele u_i's existieren (o.B.d.A. sei wieder $v \notin N(u)$ angenommen). Ansonsten setze $N(u) := o(u) - \{v\}$.

   (c) Setze

   $$V_v := \bigcup_{u \in B} N(u).$$

2. **Cocitation-Grad berechnen**

   (a) $\deg(u, w) := |i(u) \cap i(w)|$ bezeichne den *Cocitation-Grad* von u und w.

   (b) Bestimme einen Vektor $(\deg(v, u))_{u \in V_v}$ von Cocitation-Graden von v und seinen V_v-Nachbarn. Eine absteigende Ordnung der Elemente dieses Vektors liefert ein Ranking der zu v in dem Sinne verwandten Seiten, dass sie am häufigsten zusammen mit v zitiert werden.

2.26 Bemerkung In der Praxis achtet man darauf, dass der Ranking-Vektor hinreichend groß ist, z.B. kann

$$|\{u \in V_v \mid \deg(v,u) \geq 2\}| \overset{!}{\geq} 15$$

als Kriterium für eine erfolgreiche Durchführung des Algorithmus verwendet werden. Ist das Kriterium nicht erfüllt, versucht man

$$\mathrm{URL}(v) = w/w' \quad \text{mit } w \in \Sigma^*, w' \in (\Sigma - \{/\})^+$$

zu schreiben, d.h. aus der Web-Adresse URL(v) eine um eine Ebene verkürzte Adresse generieren. Der Cocitation-Algorithmus wird dann mit der verkürzten Adresse w erneut durchgeführt. (Beispiel: `http://www.foo.com/x/y/z.html` wird zu `http://www.foo.com/x/y`) Dieses Vorgehen kann man iterieren bis das Kriterium erfüllt ist oder die Rest-URL nicht weiter gekürzt werden kann. ◇

2.5 Suchmaschinen

Eine sehr gute Übersicht zu aktuellen Suchmaschinen (Suchstrategie, Aktualität des Inhalts (*freshness*, Abdeckung (des WWW), etc.)) findet sich im Skriptum zur Vorlesung *Multimedia Retrieval* von R. Weber an der ETH Zürich aus dem Sommersemester 2005 unter

`http://www.dbis.ethz.ch/education/ss2005/mmr_05/index`

Dort finden sich auch weitere Literaturhinweise zum Web-Retrieval.

2.6 Literatur

Eine grundlegende Quelle zum PageRank ist die Originalarbeit [7]. Die Grundlagenarbeit zu Hubs und Authorities ist [11]. Eine weitere interessante Arbeit von Kleinberg zur WWW-Modellierung ist [12]. Cocitation- und Companion-Algorithmus werden in einer Arbeit von Dean und Henzinger behandelt [13]. Für das ergänzende Studium sind auch einige der weiteren Arbeiten von Monika R. Henzinger zu empfehlen.
Mathematische Grundlagen zur Wahrscheinlichkeitsrechnung und zu Markovketten finden sich etwa im Lehrbuch [8]. Algorithmen zur numerischen Bestimmung von Eigenvektoren und Eigenwerten finden sich in gängigen Lehrbüchern zur praktischen bzw. numerischen Mathematik, z.B. [9] oder [14, 15]. Nicht-negative Matrizen behandelt das Lehrbuch von Seneta [10].

Kapitel 3

Musikretrieval

Partituren können als Spielanleitungen für musikalische Werke angesehen werden. Üblicherweise kann solch eine Anleitung in verbindliche und der Gestaltung des Interpretierenden überlassene Teile strukturiert werden. Verbindliche Teile sind beispielsweise im allgemeinen Tonhöhenangaben, Metrik oder Rhythmik. Das genaue Tempo, die exakte zeitliche Realisierung der Einsatzzeiten oder auch die Verwendung musikalischer Verzierungen wie beispielsweise Arpeggiaturen oder Triller, sind hingegen Teile, die allgemein der Gestaltung des Interpreten freigestellt sind. In gewisser Weise stellt eine Partitur die Darstellung eines Werkes in einer — aus musikalischer Sicht leblosen — Normalform dar.
Das MIDI-Format [16] dient demgegenüber der Vermittlung konkreter Realisierungen musikalischer Werke, wobei grob eine zeitliche Abfolge von Noten unter der Angabe von Tonhöhen und relativ fein quantisierten Einsatzzeiten und Notendauern (Übermittlung eines Endpunktes einer Note) übermittelt werden kann. Polyphonie wird hier durch die Übertragung von Zeitabständen zwischen den sequentiell übertragenen Notenereignissen realisiert.
Die in diesem Kapitel vorgestellten Algorithmen zur Suche in polyphonen Musikdaten sind prinzipiell unabhängig vom gewählten symbolischen Format verwendbar. Im folgenden soll lediglich angenommen werden, dass für jede Note eines musikalischen Werks die Parameter *Tonhöhe, Einsatzzeit* und *Tondauer* gegeben sind. Im Falle einer Partitur können physikalische Einsatzzeiten etwa durch Wahl eines konkreten Realsierungstempos erzeugt werden.

3.1 Datenmodellierung

Das Ziel von polyphonen Musikanfragen besteht darin, anhand eines gegebenen Ausschnitts einer Partitur (Anfrage) alle Stellen in einer Kollektion polyphoner Musikstücke (Datenbank) zu lokalisieren, die mit der Anfrage übereinstimmen oder dieser ähneln (Trefferbegriff). Wir betrachten in diesem Kapitel zunächst Trefferbegriffe, die über starre Zeit- und Tonhöhenverschiebungen der angefragen

Notenkonstellation spezifiziert werden können. Um in monophonen Musikstücken suchen zu können, lassen wir an späterer Stelle zusätzlich zeitvariante Deformationen der Einsatzeiten zu.

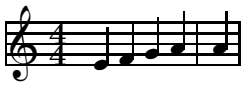


Abbildung 3.1: Beispiel einer monophonen Notenfolge.

Zunächst stellt sich die grundlegende Frage einer geeigneten Datenmodellierung. Ältere Ansätze zur Datenmodellierung ordnen einem Musikstück eine Zeichenkette zu, etwa eine Folge von Tonhöhen. Beispielsweise könnte der Notenfolge in Abb. 3.1 die Zeichenkette **efgaa** zugeordnet werden.

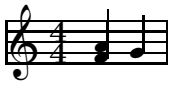


Abbildung 3.2: Beispiel eines polyphonen Musikfragments.

Bei polyphoner Musik ist dies nicht ohne weiteres möglich, bzw. nicht in solch kanonischer Weise. Welche Zeichenkette soll man beispielsweise dem Notenfragment in Abb. 3.2 zuordnen? Da die ersten beiden Noten zeitlich gleichberechtigt sind, könnte man in Mengenschreibweise etwa **{f,a}g** schreiben, was natürlich keine kanonische Codierung ist. Im vorliegenden Fall entstehen die Probleme dadurch, dass beim Übergang von den Partiturdaten zu einer Zeichenkette bereits einige Informationen weggelassen werden (z.B. Tondauer und Einsatzzeit), um eine eindimensionale Folgendarstellung zu erreichen. Bei polyphonen Musikdaten scheitert dieses Vorgehen jedoch daran, dass polyphone Musikdaten i.a. nicht eindimensional sind.
Wir betrachten nun folgenden Ansatz zur Lösung dieser Problematik:

- Anstatt Folgen von Tonhöhen zu betrachten, verwenden wir nunmehr *Mengen* elementarer Notenobjekte. Im Gegensatz zur Codierung von Noten durch alleinige Angabe der Tonhöhen, geben wir hier zu jeder Note explizit alle relevanten Parameter, und insbesondere die Einsatzzeit, an.

- Die zur Indexierung benötigten Informationen extrahieren wir anwendungsabhängig unter Beibehaltung der Struktur bzw. Objektkonstellationen.

Zur Spezifikation eines ersten Trefferbegriffs wollen wir zunächst von allen Parametern außer *Tonhöhen* und *Einsatzzeiten* abstrahieren. Wir gehen im folgenden stets davon aus, dass (Musik-) Dokumente Mengen von Noten sind. Die Menge aller möglicher Noten wird durch eine Menge U, das *Objektuniversum*, modelliert.

3.1 Definition Sei U eine Menge. Ein *Dokument* (über U) ist eine endliche Teilmenge $D \subseteq U$. Eine *Datenkollektion* (über U) ist eine Folge

$$\mathcal{D} := (D_1, \ldots, D_N)$$

von Dokumenten. ◇

Ist $\mathcal{P}$ die Menge aller zulässiger Tonhöhen ($\mathcal{P}$ von engl. *pitch*), dann setzen wir zur Modellierung von Noten

$$U := \mathbb{Z} \times \mathcal{P}$$

wobei $[t, p] \in U$ aus einer Einsatzzeit t und einer Tonhöhe p besteht. Im Falle von im MIDI-Format vorliegenden Partiturdaten wählt man entsprechend der in MIDI verfügbaren Tonhöhenwerte $\mathcal{P} := [0 : 127]$. Wir benutzen hier für $a < b$ die Konvention $[a : b] := \{a, a + 1, \ldots, b\} \subset \mathbb{Z}$. Partiturdokumente können nun als Teilmengen von $\mathbb{Z} \times \mathcal{P}$ angegeben werden.

3.2 Definition Eine *Anfrage* an eine Datenbank über einer Menge U ist eine endliche Teilmenge $Q \subseteq U$. ◇

Anfragen sind in unserem Modell also ebenfalls Dokumente. Offenbar ist es sinnvoll, nach allen Stellen innerhalb eines Musikstücks zu fragen, an denen eine gegebene Anfrage vorkommt. Eine Anfrage wird dabei meist, wie einleitend bereits skizziert, ein Notenfragment oder Ausschnitt eines Musikstücks sein. Um das Vorkommen einer Anfrage in einem Dokument und somit den Trefferbegriff spezifizieren zu können, definieren wir zunächst zeitliche Verschiebungen von Dokumenten. Für $t \in \mathbb{Z}$ sei

$$D + t := \{[t + T, p] \mid [T, p] \in D\}$$

die um t Zeiteinheiten verschobene Version des Dokuments $D \subseteq U$. Treffer sind dann Paare (t, i) aus möglichen Verschiebungen t und Dokumentennummern i, die eine Anfrage in das i-te Dokument der Datenkollektion transportieren:

3.3 Definition Sei $\mathcal{D} = (D_1, \ldots, D_N)$ eine Datenkollektion über U und $Q \subseteq U$ eine Anfrage. Die *Treffermenge* zur Anfrage Q ist definiert durch

$$H_{\mathcal{D}}(Q) := \{(t, i) \mid Q + t \subseteq D_i\}.$$

Ein Element aus $H_{\mathcal{D}}(Q)$ heißt (Q-)*Treffer*. ◇

3.4 Beispiel

1. Sei $\mathcal{P} := \{c, d, e, f\}$ die Tonhöhenmenge. Wir betrachten die Dokumente

   $$D_1 := \{[0, c], [2, e], [4, f], [5, c]\}$$
   $$D_2 := \{[10, f], [11, c]\}$$

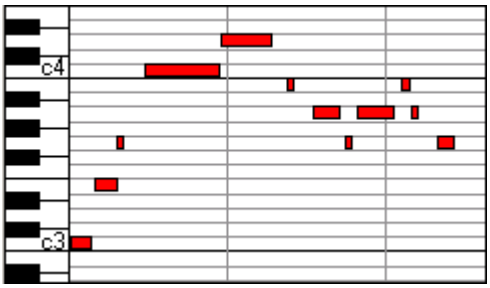


Abbildung 3.3: Anfang des Stücks *Morning has broken* in Klavierwalzendarstellung.

und die Anfrage $Q := \{[1, f], [2, c]\}$. Dann beschreiben $Q + 3 \subset D_1$ und $Q + 9 = D_2$ alle Treffer, also erhalten wir die Treffermenge

$$H_{\mathcal{D}}(Q) = \{(3, 1), (9, 2)\}.$$

2. Als weiteres Beispiel betrachten wir den Anfang des Stücks *Morning has broken* von Cat Stevens, siehe Abb. 3.3. Zur Darstellung verwenden wir hier die sogenannte *Klavierwalzen-* oder *Piano-Roll*-Darstellung. Hierbei werden die Noten eines Musikstücks als Rechtecke in der Ebene dargestellt. Die x-Achse ist die Zeitachse, während die y-Achse die Tonhöhe beschreibt. Der linke Kante des Rechtecks zur einer Note stellt dabei die Einsatzzeit, die Breite des Rechtecks die Notendauer und die untere Kante die Tonhöhe dar.

   Das zugehörige Dokument besteht aus Noten mit MIDI-Tonhöhen und in *Delta-Ticks* notierten Einsatzzeiten:

$$D_3 := \{[0, 60], [74, 64], [148, 67], [238, 72], [476, 74], [690, 71], \\ [768, 69], [872, 67], [912, 69], [1048, 71], [1084, 69], [1164, 67]\}$$

◇

3.2 Effiziente Trefferberechnung

Unser Ziel ist nun, einen Algorithmus zur schnellen Bestimmung der Treffermenge $H_{\mathcal{D}}(Q)$ zu gegebener Anfrage $Q \subseteq U$ anzugeben. Dazu benötigen wir folgendes Resultat, das es uns erlaubt zur Trefferermittlung eine Divide-and-Conquer Strategie zu verwenden.

3.5 Lemma Seien $A, B \subseteq U$, dann gilt

$$H_{\mathcal{D}}(A \cup B) = H_{\mathcal{D}}(A) \cap H_{\mathcal{D}}(B).$$

◇

Beweis: Die Behauptung folgt aus

$$\begin{aligned}(t,i) \in H_{\mathcal{D}}(A \cup B) &\Leftrightarrow (A \cup B) + t \subseteq D_i \\ &\Leftrightarrow (A + t \subseteq D_i) \wedge (B + t \subseteq D_i) \\ &\Leftrightarrow (t,i) \in [H_{\mathcal{D}}(A) \cap H_{\mathcal{D}}(B)].\end{aligned}$$

□

Mit der Schreibweise $H_{\mathcal{D}}(q) := H_{\mathcal{D}}(\{q\})$ für $q \in U$ können wir nun die Bestimmung der Treffermenge als Schnitt von, bezüglich der einzelnen Noten der Anfrage spezifizierten, Mengen vornehmen.

3.6 Korollar Sei $Q \subseteq U$ eine Anfrage, dann gilt

$$H_{\mathcal{D}}(Q) = \bigcap_{q \in Q} H_{\mathcal{D}}(q).$$

◇

Haben wir also für beliebiges $q \in U$ Zugriff auf die Menge $H_{\mathcal{D}}(q)$, so können wir für beliebige Anfragen Q die Menge aller Q-Treffer bestimmen.
Ein Problem besteht nun noch darin, dass U wie in unserem Beispiel unendlich groß sein kann, so dass nicht alle Mengen $H_{\mathcal{D}}(q)$ im Index abgelegt werden können. Folgendes Resultat besagt, dass dies auch gar nicht notwendig ist, da sich alle diese Mengen zu einer festen Tonhöhe auseinander bestimmen lassen.

3.7 Lemma Für $\ell \in \mathbb{Z}$ und eine Teilmenge $Q \subseteq U$ definiere

$$H_{\mathcal{D}}(Q) + \ell := \{(t + \ell, i) \mid (t,i) \in H_{\mathcal{D}}(Q)\},$$

dann gilt

$$H_{\mathcal{D}}([a+b,p]) = H_{\mathcal{D}}([a,p]) - b.$$

◇

Beweis: Die Behauptung folgt wegen

$$\begin{aligned}(t,i) \in H_{\mathcal{D}}([a+b,p]) &\Leftrightarrow t + [a+b,p] \in D_i \\ &\Leftrightarrow t + b + [a,p] \in D_i \\ &\Leftrightarrow (t+b,i) \in H_{\mathcal{D}}([a,p]) \\ &\Leftrightarrow (t,i) \in H_{\mathcal{D}}([a,p]) - b.\end{aligned}$$

□

Insbesondere folgt für $p \in \mathcal{P}$: $H_{\mathcal{D}}([t,p]) = H_{\mathcal{D}}([0,p]) - t$. Für eine Menge $\mathcal{P}$ von Tonhöhen reicht es also beispielsweise aus, nur die Mengen $H_{\mathcal{D}}([0,p])$ für alle $p \in \mathcal{P}$ abzuspeichern. Ist $\mathcal{P}$ endlich, ist dies eine endliche Anzahl von Mengen. Diese Mengen bilden den *Suchindex* $H_{\mathcal{D}}([0,p])_{p \in \mathcal{P}}$.
Fassen wir die letzten beiden Resultate zusammen, erhalten wir folgendes wichtige Resultat zur Charakterisierung der Treffermenge mit Hilfe von Grundmengen der Form $H_{\mathcal{D}}([0,p])$.

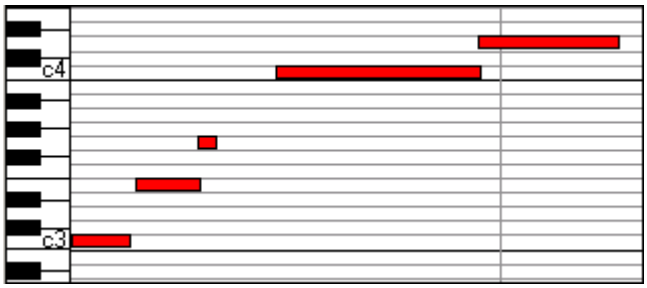


Abbildung 3.4: Anfrage bestehend aus einem Teil des Anfangs von *Morning has broken*

3.8 Satz Sei $Q \subseteq U$ eine Anfrage, dann lässt sich die Treffermenge durch die Schnittmenge

$$H_{\mathcal{D}}(Q) = \bigcap_{[t,p]\in Q} H_{\mathcal{D}}([0,p]) - t \tag{3.1}$$

bestimmen. ◇

Dieser Satz folgt aus dem Lemma 3.7 und Korollar 3.6. In Fortsetzung des Beispiels 3.4 auf Seite 106 erhalten wir:

$$H_{\mathcal{D}}([0,c]) = \{(0,1),(5,1),(11,2)\},$$

$$H_{\mathcal{D}}([0,e]) = \{(2,1)\},$$

$$H_{\mathcal{D}}([0,f]) = \{(4,1),(10,2)\}$$

und

$$\begin{aligned} H_{\mathcal{D}}(Q) &= (H_{\mathcal{D}}([0,f]) - 1) \cap (H_{\mathcal{D}}([0,c]) - 2) \\ &= \{(3,1),(9,2)\} \cap \{(-2,1),(3,1),(9,2)\} \\ &= \{(3,1),(9,2)\} \end{aligned}$$

in Übereinstimmung mit der bereits oben angegebenen Treffermenge.
Als weiteres Beispiel für eine Anfrage betrachten wir den in Abbildung 3.4 dargestellten Teil des Anfangs aus *Morning has broken*. Die entsprechenden Noten sind $Q' = \{[0,60],[74,64],[148,67],[238,72],[476,74]\}$. Da $Q' + 0 \subset D_3$, ist $(0,3)$ ein Treffer. In Abb. 3.5 ist das Vorkommen von Q' im obigen Dokument D_3 hervorgehoben.
Versieht man $\mathbb{Z}\times\mathcal{P}$ mit einer Totalordnung, so können die Mengen $H_{\mathcal{D}}(A), A \subseteq U$ *sortiert* abgelegt werden. Insbesondere wird man die als Index verwendeten Mengen $H_{\mathcal{D}}([0,p])$ als sortierte *Listen* ablegen. Diese Listen enthalten die Stellen aller Vorkommen der Note p. Dies erinnert an die Technik eines invertierten Indexes (ähnlich z.B. dem Stichwortverzeichnis eines Buches). Daher nennen wir diese

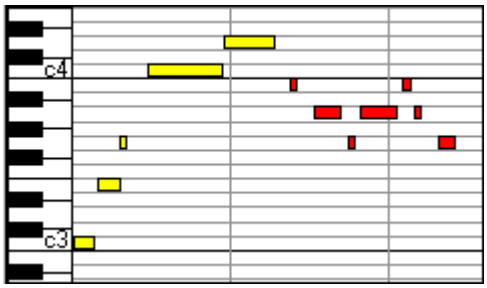


Abbildung 3.5: Vorkommen von Q' (hervorgehoben) im Dokument D_3.

Listen — in Anlehnung an den entsprechenden Begriff aus der Volltextsuche [4] — auch *invertierte Listen*. Der Index wird manchmal auch *invertierter Index* genannt.
Sind die Listen sortiert, so kann die Schnittmengenbildung aus (3.1) durch Verschmelzungen (Merges) sortierter Listen durchgeführt werden. Dies geht für eine Anfrage $Q := \{[t_1, p_1], \ldots, [t_n, p_n]\}$ mit Listenlängen $\ell_i := |H_{\mathcal{D}}([0, p_i])|$ sehr effizient. Die Elemente aus Q seien nach Listenlängen aufsteigend nummeriert, d.h. $\ell_1 \leq \ldots \leq \ell_n$. Das Verschmelzen der in jedem Schritt sortiert vorliegenden Listen benötigt unter Verwendung von binärer Suche $O(n\ell_1 \log \ell_n)$ Schritte. Sind die in einem Schritt zu verschmelzenden Listen von nicht zu unterschiedlichen Längen s und t, kann ein lineares Verschmelzen in $O(s + t)$ Schritten schneller sein. Setzt man $\ell := \ell_1 + \ldots + \ell_n$, kann als weitere Alternative mittels n-Wege Merge eine Laufzeit von $O(\ell \log n)$ erreicht werden. Hierzu bildet man zunächst aus den jeweils kleinsten Elementen jeder Liste in $O(n)$ Schritten einen Heap. Nun entfernt man sukzessive das kleinste Element und lässt das nächstkleinste aus dem Heap nachsickern, wobei jeweils die nächsten Elemente der sortierten Listen nachrücken (insgesamt $\ell \cdot \log n$ Schritte). Ein Element ist genau dann im Schnitt, wenn es während dieses Vorgangs n aufeinanderfolgende Male kleinstes Element des Heaps ist.

3.3 Konzepte zur fehlertoleranten Suche

Das Erzielen von Treffern wie im letzten Abschnitt beschrieben erfordert, dass die Anfrage die präzise Angabe der Konstellation der Notenobjekte an der Trefferposition enthält. Dieses Wissen ist jedoch oft nicht vorhanden. So kommt es zu Unterschieden zwischen angefragtem und gesuchtem Dokument. Diese Unterschiede sind strenggenommen nicht immer *Fehler* des Anfragenden — so gibt es z.B. häufig verschiedene bekannte Versionen eines Musikstücks. Wir wollen trotzdem im folgenden von *fehlertoleranter* Suche sprechen und betrachten zunächst einige Beispiele für Fälle, die Fehlertoleranz erfordern:

- Eine Anfrage Q aus $|Q| = n$ Noten enthält $k < n$ Noten, die nicht im Trefferdokument vorkommen, obwohl alle anderen $n-k$ Noten vorkommen.
- Die Anfragekonstellation weicht leicht von einer Stelle in einem Dokument ab, z.B.
  - durch leichte Unterschiede zwischen den Abständen der Einsatzzeiten,
  - durch eine abweichende Tonhöhe eines Tons, resultierend aus einem fehlerhaften Tonhöhenintervall.
- An gewissen Stellen der Anfrage ist unklar, welche Note genau an der gesuchten Trefferposition vorkommt, obwohl die Anzahl möglicher Noten durch eine kleine Anzahl möglicher Alternativen eingeschränkt ist.

Die in diesem Abschnitt vorgestellten Konzepte für eine fehlertolerante Suche erlauben in solchen und weiteren Fällen, trotzdem Treffer zu erhalten. Dabei ist es teilweise nötig, den Trefferbegriff, das zugrundeliegende Universum von Elementarobjekten, und bestehende Suchalgorithmen zu erweitern.

3.3.1 Toleranz gegenüber Fehlstellen

In Anlehnung an das erste obige Beispiel definieren wir für $Q \subseteq U$ und $k \in \mathbb{N}$ die Menge

$$H_{\mathcal{D},k}(Q) := \{(t, i) \mid |(t + Q) \backslash D_i| \leq k\}$$

aller *Treffer mit höchstens k Fehlstellen* (k-Mismatch-Treffer).
Abb. 3.6 illustriert das Konzept der Fehlstellen. Eine Anfrage (oben) kommt bis auf zwei Fehlstellen in einem Dokument (mitte) vor. Das Vorkommen ist im unteren Teil der Abbildung dargestellt. Die Fehlstellen und die übereinstimmenden Noten sind unterschiedlich markiert.

3.9 Beispiel In Fortsetzung des Beispiels zum Stück *Morning has broken* aus dem letzten Abschnitt zeigt Abb. 3.7 ein Vorkommen der dort definierten Anfrage Q' bis auf eine Fehlstelle in einem Stück von Mozart. In diesem Beispiel wurde die letzte Note $[476, 74]$ aus Q' nicht gematcht.

Die Berechnung von $H_{\mathcal{D},k}(Q)$ kann effizient mit einem auf dynamischer Programmierung basierenden Algorithmus geschehen. Hierzu sei $Q =: \{q_1, \ldots, q_n\}$ eine beliebige Nummerierung der Elemente aus Q. Definiere $H_j := H_{\mathcal{D}}(q_j)$ als Liste zum j-ten Element. Weiterhin sei

$$\Gamma_j := H_1 \cup \ldots \cup H_j$$

die Menge aller Trefferkandidaten bis zum j-ten Schritt des Algorithmus. Für jeden Schritt des Algorithmus definieren wir nun eine *Kreditfunktion* $C_j : \Gamma_j \to \mathbb{Z}$. Es wird sich herausstellen, dass die Elemente $\gamma \in \Gamma_n$ mit $C_n(\gamma) > 0$ genau die Treffer mit maximal k Fehlstellen sind.

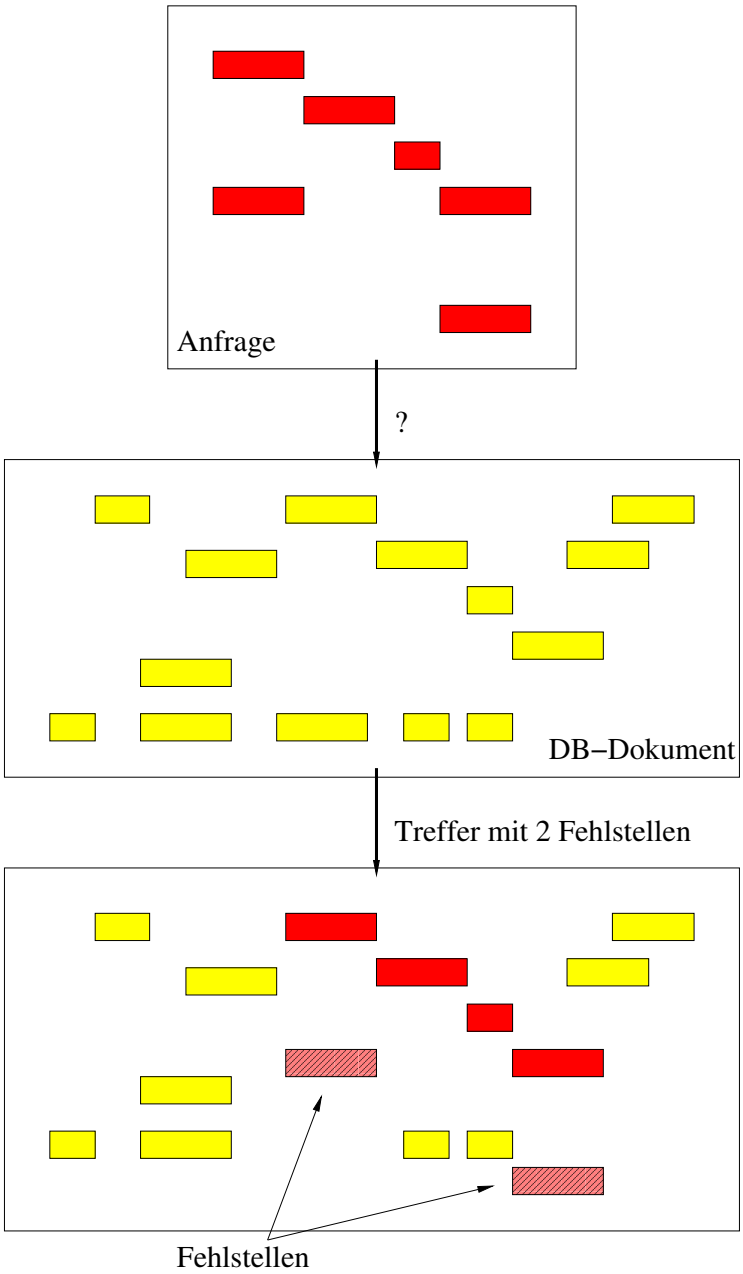


Abbildung 3.6: Anfrage (oben) liefert einen Treffer (unten) bis auf zwei Fehlstellen in einem Dokument (mitte).

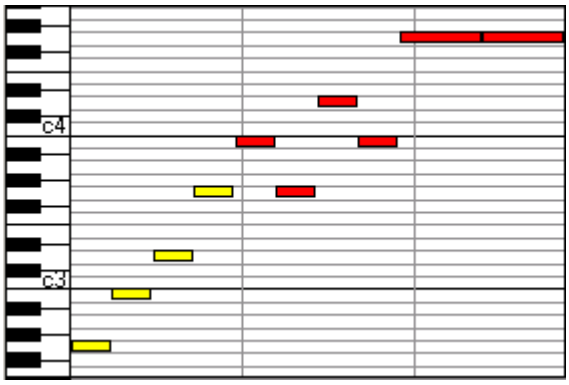


Abbildung 3.7: Vorkommen von Q' (hervorgehoben) bis auf eine Fehlstelle in einem Stück von Mozart, KV 110.

3.10 Definition (Kreditfunktionen) Für $\gamma \in \Gamma_1$ setze $C_1(\gamma) := k+1$. Für $2 \leq j \leq n$ definiere induktiv

$$C_j(\gamma) := \begin{cases} C_{j-1}(\gamma) & \text{falls } \gamma \in \Gamma_{j-1} \cap H_j, \\ C_{j-1}(\gamma) - 1 & \text{falls } \gamma \in \Gamma_{j-1} \backslash H_j, \\ k+2-j & \text{falls } \gamma \in H_j \backslash \Gamma_{j-1}. \end{cases}$$

◇

3.11 Satz Es gilt

$$H_{\mathcal{D},k}(Q) = \{\gamma \in \Gamma_n \mid C_n(\gamma) > 0\}.$$

◇

Beweis: Sei $(t,i) \in \Gamma_j$ und $Q_j := \{q_1, \ldots, q_j\}$. Wir zeigen $C_j(t,i) = k+1-|(t+Q_j) \setminus D_i|$ per Induktion nach j. Für $j=1$ gilt dies nach Konstruktion von Γ_1 und Q_1 offensichtlich. Für den Induktionsschritt $(j-1 \to j)$ unterscheiden wir drei Fälle:
Fall 1: $(t,i) \in \Gamma_{j-1} \cap H_j$. Da $t+q_j \in D_i$, ist $(t+Q_j) \setminus D_i = (t+Q_{j-1}) \setminus D_i$. Daher folgt $C_j(t,i) := C_{j-1}(t,i) = k+1-|(t+Q_{j-1}) \setminus D_i| = k+1-|(t+Q_j) \setminus D_i|$.
Fall 2: $(t,i) \in \Gamma_{j-1} \setminus H_j$. Daher folgt $t+q_j \notin D_i$, also $|(t+Q_j) \setminus D_i| = |(t+Q_{j-1}) \setminus D_i| + 1$ und daher $C_j(t,i) := C_{j-1}(t,i) - 1 = k+1-(|(t+Q_{j-1}) \setminus D_i|+1) = k+1-|(t+Q_j) \setminus D_i|$.
Fall 3: $(t,i) \in H_j \backslash \Gamma_{j-1}$. Dann ist $t+q_j \in D_i$ aber $t+q_\ell \notin D_i$, für alle $\ell \in [1:j-1]$. Daher gilt $|(t+Q_j) \setminus D_i| = j-1$, also $C_j(t,i) := k+2-j = k+1-(j-1) = k+1-|(t+Q_j) \setminus D_i|$.
Da $Q_n = Q$, erhält man $C_n(t,i) = k+1-|(t+Q) \setminus D_i|$ für jedes $(t,i) \in \Gamma_n$. Also gilt $C_n(t,i) > 0$ gdw. $|(t+Q) \setminus D_i| \leq k$ bzw. $(g,i) \in H_{\mathcal{D},k}(Q)$. □
Zur Bestimmung von $H_{\mathcal{D},k}(Q)$ bearbeitet man wieder alle benötigten Listen aufsteigend der Größe nach. Wegen $\gamma \in H_{\mathcal{D},k}(Q) \Rightarrow \gamma \in \Gamma_{k+1}$ folgt $H_{\mathcal{D},k}(Q) \subseteq \Gamma_{k+1}$.

Verwendet man wie im Falle der Bestimmung der exakten Treffer zum Verschmelzen der Listen binäre Suche, so erhält man eine Aufwandsabschätzung von $O(|\Gamma_n| \cdot |Q| \cdot \log \ell_n)$ Schritten zum Verschmelzen der Listen und zur Berechnung der Treffermenge $H_{\mathcal{D},k}(Q)$. Diese Laufzeit kann man aber meist signifikant beschleunigen, indem man bei jedem Berechnungschritt bereits diejenigen Zwischenergebnisse mit nicht-positivem Kredit weglässt. Dazu definieren wir $\Gamma^1 := \Gamma_1$ und für $j \geq 2$

$$\Gamma^j := \{\gamma \in (\Gamma^{j-1} \cup H_j) \mid C^j(\gamma) > 0\}.$$

Hierbei definieren wir wie oben Kreditfunktionen $C^1(\gamma) := k + 1$ und für $j \geq 2$

$$C^j(\gamma) := \begin{cases} C^{j-1}(\gamma) & \text{falls } \gamma \in \Gamma^{j-1} \cap H_j, \\ C^{j-1}(\gamma) - 1 & \text{falls } \gamma \in \Gamma^{j-1} \backslash H_j, \\ k + 2 - j & \text{falls } \gamma \in H_j \backslash \Gamma^{j-1}, \end{cases}$$

wobei $C^j : \Gamma^{j-1} \cup H_j \to \mathbb{Z}$.
Offensichtlich gelten $\Gamma^j \subseteq \Gamma^{j-1} \cup H_j \subseteq \Gamma_j$ sowie $\gamma \in H_{\mathcal{D},k}(Q) \Rightarrow \gamma \in \Gamma^k$. Weiterhin gilt $\forall \ell \geq k$ die Inklusion $\Gamma^\ell \supseteq \Gamma^{\ell+1}$, d.h. ab dem k-ten Schritt wird die Menge der Trefferkandidaten nicht größer. Somit erhalten wir die verbesserte Laufzeit von $O(|\Gamma^k| \cdot |Q| \cdot \log \ell_n)$ Schritten zur Bestimmung von Γ^n. Man zeigt mit Hilfe der letztgenannten Eigenschaften leicht, dass $H_{\mathcal{D},k}(Q) = \Gamma^n$ gilt.
Verwendet man alternativ die in Abschnitt 3.2 skizzierte n-Wege Merge-Technik, erhält man eine Laufzeit von $O(\ell \log n)$ Schritten zur Ermittlung von $H_{\mathcal{D},k}(Q)$, wobei $\ell := \ell_1 + \ldots + \ell_n$ mit $\ell_i := |H_{\mathcal{D}}(q_i)|$ wie oben definiert ist. Die Merge-Technik wird dabei dahingehend modifiziert, dass alle diejenigen Elemente als Treffer ausgegeben werden, die bei der Abarbeitung des Heaps mindestens $n - k$-Mal hintereinander als kleinstes Element vom Heap entfernt werden.

3.3.2 Fuzzy–Suche

Der in diesem Abschnitt behandelten unscharfen- oder Fuzzy-Suche liegt die Idee zugrunde, dass es manchmal innerhalb einer Anfrage erlaubt sein soll, anstelle einer bestimmten Note $q \in U$ vielmehr eine Menge $Q \subset U$ von Alternativen zu spezifizieren. Dies modelliert den Fall, dass ein Anfragender nicht genau weiß, welche dieser Alternativen an der gesuchten Trefferstelle vorkommt.
Das Konzept der Fuzzy-Suche ist in Abbildung 3.8 illustriert. Die Anfrage (oben) enthält zwei Fuzzy-Noten, eine bestehend aus zwei, die andere aus drei Alternativen. Die Anfrage liefert einen Treffer (unten) in einem Dokument (mitte), wobei jeweils eine der Fuzzy-Noten im Trefferdokument matcht. Die gematchten Noten sind im Trefferdokument hervorgehoben, wobei die Fuzzy-Noten unterschiedlich markiert sind.
Eine *Fuzzy-Anfrage* ist definiert als eine Folge $\mathbf{F} := (F_1, \ldots, F_n)$ aus n endlichen Alternativenmengen $F_i \subset U$. Einer Fuzzy-Anfrage kann man durch

$$Q(\mathbf{F}) := \{\{q_1, \ldots, q_n\} \mid \forall i : q_i \in F_i\}$$

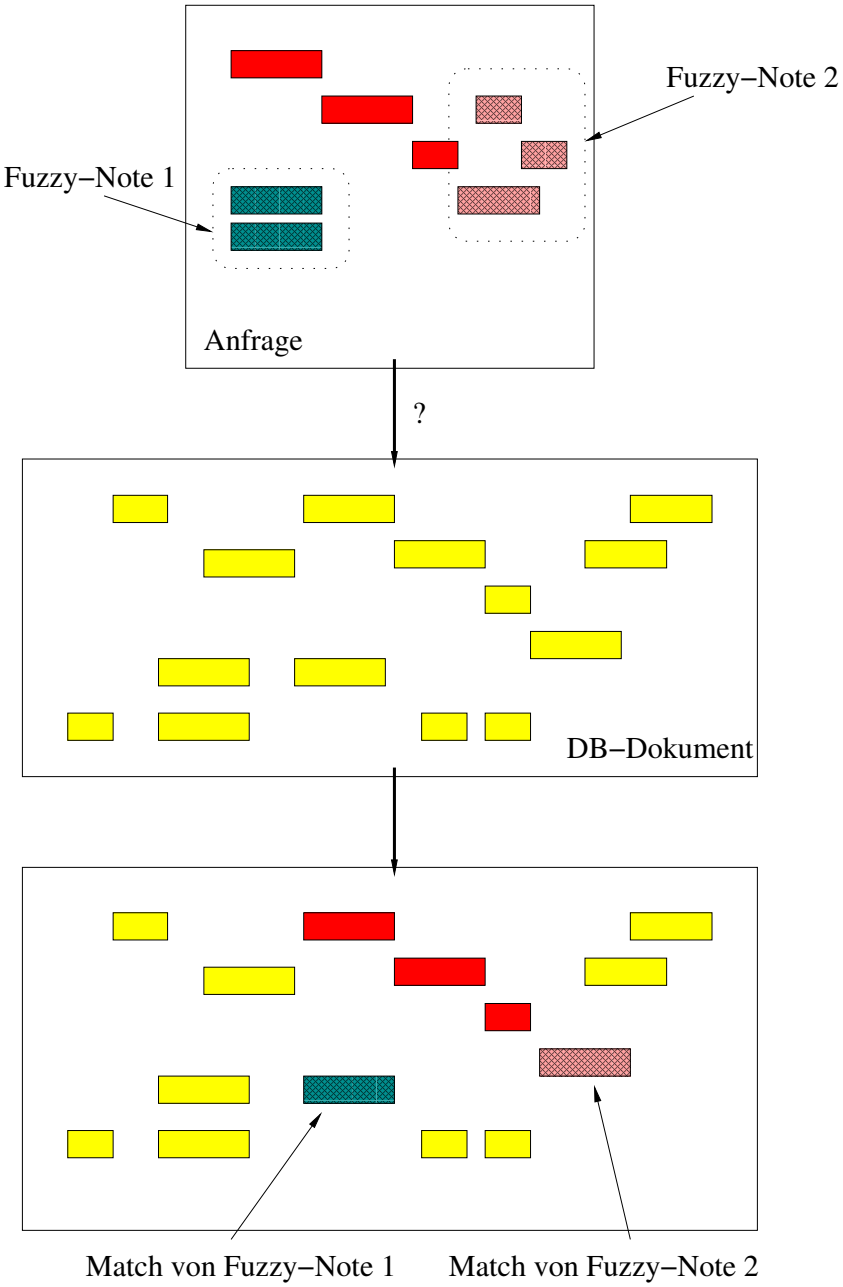


Abbildung 3.8: Anfrage (oben) mit zwei Fuzzy-Noten und Trefferstelle (unten) in einem Dokument (mitte).

eine Familie zugehöriger Elementaranfragen zuordnen. Jede dieser Elementaranfragen enthält genau ein Element jeder der Alternativenmengen aus $\mathbf{F}$. Fasst man die Treffermengen zu allen diesen Elementaranfragen zusammen, erhält man auf natürliche Weise die Menge

$$H_{\mathcal{D}}(\mathbf{F}) := \{(t,i) \mid \exists Q \in Q(\mathbf{F}) : t + Q \subseteq D_i\}$$

aller Treffer zur Fuzzy-Anfrage $\mathbf{F}$. Sind die Alternativenmengen F_i paarweise disjunkt, induziert $\mathbf{F}$ offenbar $\prod_{i=1}^{n} |F_i|$ Elementaranfragen. Eine naive Bestimmung von $H_{\mathcal{D}}(\mathbf{F})$ durch Berechnung der Treffermengen zu allen Elementaranfragen ist somit i.a. ineffizient. Folgendes Resultat erlaubt eine effizientere Bestimmung aller Fuzzy-Treffer unter Verwendung von Mengenschnitten und -vereinigungen:

3.12 Satz Sei $\mathcal{D} = (D_1, \ldots, D_N)$ mit $D_i \subseteq U$ Kollektion von Musikdokumenten. Ist $\mathbf{F} := (F_1, \ldots, F_n)$ eine Fuzzy-Anfrage aus n Alternativenmengen $F_i \subset U$, so gilt

$$H_{\mathcal{D}}(\mathbf{F}) = \bigcap_{j \in [1:n]} \left(\bigcup_{q \in F_j} H_{\mathcal{D}}(q) \right). \tag{3.2}$$

◇

Beweis: $(t,i) \in H_{\mathcal{D}}(\mathbf{F})$ ist äquivalent zu $(t+Q) \subseteq D_i$, für ein $Q \in Q(\mathbf{F})$. Dies ist gleichbedeutend mit der Existenz eines $q_j \in F_j$ mit $t + q_j \in D_i$, für jedes $j \in [1:n]$, d.h. $(t,i) \in \cup_{q_j \in F_j} H_{\mathcal{D}}(q_j)$, für alle j, woraus die Behauptung folgt.

□

Es ist anzumerken, dass die Schnittbildung (3.2) für einen Treffer $(t,i) \in H_{\mathcal{D}}(\mathbf{F})$ keine Information über den zugehörigen Elementartreffer $Q \in Q(\mathbf{F})$ mit $Q + t \subseteq D_i$ liefert. Wird eine solche Information gewünscht, können die entsprechenden Daten etwa bei der Trefferbestimmung geeignet für jeden Trefferkandidaten mitprotokolliert werden.

Obwohl $\mathbf{F}$ bei paarweise disjunkten F_i-Mengen $\prod_{i=1}^{n} |F_i|$ verschiedene Elementaranfragen induziert, ist der Aufwand zur Bestimmung von $H_{\mathcal{D}}(\mathbf{F})$ nicht multiplikativ in den $|F_i|$. Die Trefferbestimmung ist wegen (3.2) wie bereits gesehen möglich in $O(n\ell'_1 \log \ell'_n)$ Schritten wobei eine aufsteigende Folge $\ell'_1 \leq \ldots \leq \ell'_n$ von Listenlängen

$$\ell'_i := \left| \bigcup_{q \in F_i} H_{\mathcal{D}}(q) \right|$$

zugrunde gelegt wurde, wobei die n Listen gemäß (3.2) durch Vereinigung der den Alternativenmengen zugeordneten Listen entstehen. Die oben beschriebene Trefferbestimmung mittels n-Wege Merging ist entsprechend anwendbar.

3.4 Literatur

Das Grundkonzept zur Suche in polyphoner Musik wurde innerhalb der ersten Phase des MiDiLiB-Projekts, das in der AG Clausen in den Jahren 1998–2003 durchgeführt wurde, entworfen und ist in der Arbeit [17] zusammengefasst. In weiterentwickelter Sichtweise sind die Resultate zur Musiksuche in den Publikationen [18, 19] erschienen. Erweiterte, zur Melodiesuche taugliche Algorithmen wurden hierauf aufbauend von F. Kurth, A. Ribbrock und M. Clausen entwickelt [20]. Eine ausführliche Darstellung hierzu, sowie eine Übersicht zu weiteren Arbeiten in den Bereichen polyphoner Musiksuche und Melodiesuche findet sich in der auf den Webseiten der AG Clausen verfügbaren Habilitationsschrift von F. Kurth [21].

Kapitel 4

Ein allgemeines Konzept zum Multimediaretrieval

4.1 Gruppen und deren Operationen auf Mengen

In diesem Kapitel wird ein allgemeines Konzept zum Multimediaretrieval basierend auf Gruppenoperationen vorgestellt. Hierzu erinnern wir zunächst an Gruppen und das Konzept der Operation einer Gruppe auf einer Menge. Dies ermöglicht uns die Formulierung spezifischer Trefferbegriffe.

4.1 Definition Eine *Gruppe* $(G, *)$ ist eine Menge G mit einer Verknüpfung $* : G \times G \to G$, so dass

1. $\forall a, b, c \in G$ das Assoziativgesetz $(a * b) * c = a * (b * c)$ gilt,
2. ein neutrales Element $1 = 1_G$ existiert, so dass $\forall g \in G : 1 * g = g * 1 = g$ gilt,
3. $\forall g \in G$ ein inverses Element $g^{-1} \in G$ existiert mit $g * g^{-1} = 1$.

Eine Gruppe heißt *abelsch*, wenn zudem $\forall a, b \in G$ das Kommutativgesetz $a * b = b * a$ gilt. $\diamond$

4.2 Beispiel

- Wir betrachten die Gruppe $(\mathbb{Z}, +)$ der Menge aller ganzer Zahlen mit der Addition als Verknüpfung. Zu $a \in \mathbb{Z}$ ist $-a$ das inverse Element, 0 ist das neutrale Element. $(\mathbb{Z}, +)$ ist abelsch.
- Die Menge $\mathrm{GL}(n, \mathbb{R})$ aller invertierbaren $n \times n$-Matrizen mit Einträgen in $\mathbb{R}$ zusammen mit der Standardmatrixmultiplikation ist für $n \geq 2$ eine nichtabelsche Gruppe. Matrixmultiplikation ist i.a. nicht kommutativ, z.B. gilt

$$\begin{pmatrix} & 1 \\ 1 & \end{pmatrix} \begin{pmatrix} a & \\ & b \end{pmatrix} = \begin{pmatrix} & b \\ a & \end{pmatrix} \text{ aber } \begin{pmatrix} a & \\ & b \end{pmatrix} \begin{pmatrix} & 1 \\ 1 & \end{pmatrix} = \begin{pmatrix} & a \\ b & \end{pmatrix}.$$

- Die Menge S_M aller Permutationen einer endlichen Menge M ist zusammen mit der Komposition von Abbildungen eine Gruppe, die sogenannte *symmetrische Gruppe*. S_M ist für $|M| \geq 3$ nicht abelsch.

- Die n-elementige zyklische Gruppe $C_n := ([0 : n-1], +)$ mit Addition modulo n ist abelsch.

- Für n-D-Anwendungen interessant ist die *Euklidische Gruppe* aller starren Bewegungen im $\mathbb{R}^n$. Diese Gruppe wird erzeugt durch alle Translationen, Spiegelungen und Drehungen um der Ursprung.

◇

Wie in den Beispielen gesehen, schreibt man die Verknüpfung einer Gruppe manchmal multiplikativ und machmal additiv. Wir wollen im folgenden die multiplikative Version verwenden und schreiben dann statt $g * h$ meist einfach gh. Ist die Verknüpfung der Gruppe aus dem Zusammenhang klar, spricht man vereinfachend von der Gruppe G.
Oft sind auch Teilmengen von Gruppen interessant, die ihrerseits wieder Gruppen bilden:

4.3 Definition $U \subseteq G$ heißt *Untergruppe* von G, $U \leq G$, falls $1_G \in U$, $\forall u, v \in U$ stets $u * v \in U$ gilt sowie $\forall u \in U$ stets $u^{-1} \in U$ ist. ◇

4.4 Beispiel

- Für $\ell \in \mathbb{N}_{\geq 1}$ ist $(\ell\mathbb{Z}, +) \leq (\mathbb{Z}, +)$, wobei

$$\ell\mathbb{Z} := \{\ell k \mid k \in \mathbb{Z}\}.$$

- Für eine fest gewählte Dimension n ist die Gruppe aller Drehungen um den Ursprung eine Untergruppe der Euklidischen Gruppe des $\mathbb{R}^n$.

◇

Wir kommen nun zur Definition von Gruppenoperationen auf Mengen.

4.5 Definition Sei M eine Menge und G eine Gruppe. M wird zur *G-Menge* vermöge einer Abbildung

$$G \times M \to M$$

$$(g, m) \mapsto gm$$

für die folgende Gesetze gelten:

1. $\forall g, h \in G : \forall m \in M : \; g(hm) = (gh)m$ und
2. $\forall m \in M : \; 1_G m = m$.

Man sagt dann, *G operiert (von links) auf M* und spricht von einer *G-Operation auf M*. ◇

Offenbar kann man bestimmte Elemente von M durch G-Multiplikation ineinander verschieben. Genauer heißen $m, m' \in M$ *G-äquivalent*,

$$m \sim_G m', \text{ gdw. } \exists g \in G : gm = m'.$$

4.6 Lemma $(M \times M, \sim_G)$ ist eine Äquivalenzrelation. ◇

Beweis:

- $m \sim_G m$, da $1_G m = m$.
- $m \sim_G m' \Leftrightarrow \exists g \in G : gm = m' \Leftrightarrow m = g^{-1}m' \Leftrightarrow m' \sim_G m$.
- $m \sim_G n$ und $n \sim_G k$, also

$$\exists g, h \in G : gm = n \wedge hn = k \Rightarrow (gh)m = k \Rightarrow m \sim_G k.$$

□

Die Mengen aller G-äquivalenten Elemente heißen *Bahnen*.

$$Gm := \{gm \mid g \in G\}$$

heißt G-Bahn von $m \in M$.
Da jedes $m \in M$ in einer G-Bahn enthalten ist, gibt es somit eine disjunkte Zerlegung

$$M = \bigsqcup_{r \in R} Gr$$

falls $R \subseteq M$ eine Menge bestehend aus genau einem Repräsentanten pro G-Bahn ist. Gibt es nur eine G-Bahn, so heißt die Gruppenoperation *transitiv*, sonst *intransitiv*.

4.7 Beispiel

- $(\mathbb{Z} \times \mathbb{Z}, +)$ operiert wie oben gesehen per komponentenweiser Addition auf $\mathbb{Z} \times \mathbb{Z} \times \mathbb{R}_{>0}$. Diese Operation ist intransitiv. Ein Repräsentantensystem der $(\mathbb{Z} \times \mathbb{Z}, +)$-Bahnen ist
$$\{[0, 0, d] \mid d \in \mathbb{R}_{>0}\}.$$
- $(\mathbb{Z}, +)$ operiert auf dem Notenuniversum $\mathbb{Z} \times \mathcal{P}$ durch (Zeit-) Verschiebung bzw. Addition in der ersten Komponente. Ein Repräsentantensystem der $(\mathbb{Z}, +)$-Bahnen, das wir bereits kennen, ist
$$\{[0, p] \mid p \in \mathcal{P}\}.$$
- $(\mathbb{Z}, +)$ operiert auf sich selbst transitiv durch Addition.

- $(\ell\mathbb{Z}, +)$ operiert für festes $\ell \in \mathbb{N}_{>1}$ durch Addition auf $\mathbb{Z}$. Die Operation ist intransitiv. Ein Repräsentantensystem der Bahnen ist

$$R_\ell = \{k \mid 0 \leq k < \ell\}.$$

  Die Bahnen sind für $k \in R_\ell$ gegeben durch

$$k + \ell\mathbb{Z} = \{k + \ell n \mid n \in \mathbb{Z}\}.$$

- Die symmetrische Gruppe S_n operiert auf der Menge $P_k(n)$ aller k-elementigen Teilmengen X von $[1 : n]$ vermöge

$$(\pi, X) \mapsto \pi X := \{\pi(x) \mid x \in X\}.$$

  Diese S_n-Operation ist transitiv, d.h. es gibt nur eine S_n-Bahn (Übung!).

◇

Operiert G auf M, so kann es passieren, dass $m \in M$ außer vom Einselement 1_G noch von anderen $g \in G$ festgelassen wird, $gm = m$. Da

$$gm = m \Leftrightarrow m = g^{-1}m$$

und

$$gm = m \wedge hm = m \Rightarrow ghm = m$$

bilden die Elemente, die m fest lassen eine Untergruppe von G. Formal definieren wir zu $m \in M$ den *Stabilisator*

$$G_m := \{g \in G \mid gm = m\},$$

manchmal auch als *Stabilisatoruntergruppe* bezeichnet. Stabilisatoren spielen im Zusammenhang mit der Indexierung oft eine wichtige Rolle.
Wir stellen noch einige wichtige Fakten über Stabilisatoren zusammen. Ist $U < G$ Untergruppe, so operiert U auf G durch Rechtsmultiplikation

$$(u, g) \mapsto ug.$$

Die Bahnen sind die sogenannten *Rechtsnebenklassen* Ug. Ist $R \subset G$ ein Repräsentantensystem aus genau einem Element für jede Rechtsnebenklasse, dann läßt sich G disjunkt in Rechtsnebenklassen zerlegen,

$$G = \bigsqcup_{r \in R} Ur.$$

Analog kann man *Linksnebenklassen* definieren und eine weitere disjunkte Zerlegung von G angeben.

4.8 Beispiel Betrachte wie oben $\ell\mathbb{Z} < \mathbb{Z}$ und

$$\mathbb{Z} = \bigsqcup_{k \in R_\ell} \ell\mathbb{Z} + k = \bigsqcup_{k \in [0:\ell-1]} \ell\mathbb{Z} + k$$

ist disjunkte Zerlegung von $\mathbb{Z}$ in Linksnebenklassen. $\diamond$

Die Anzahl der Nebenklassen einer Untergruppe $U < G$ heißt *Index von U in G* und wird mit

$$[G : U]$$

bezeichnet. Da alle Nebenklassen (Linksnebenklassen gU und Rechtsnebenklassen Ug) gleichmächtig sind, folgt für endliche Gruppen die wichtige Identität

$$\begin{aligned} & [G : U] \cdot |U| = |G| \\ \Leftrightarrow & \\ & [G : U] = |G|/|U|. \end{aligned}$$

Der resultierende Satz von Lagrange ist grundlegend für mögliche Größen von Untergruppen:

4.9 Satz Ist U Untergruppe einer endlichen Gruppe G, dann teilt die Ordnung von U die Ordnung von G, also $|U|$ teilt $|G|$. $\diamond$

Die Menge der Linksnebenklassen von $U < G$ wird häufig auch als G/U bezeichnet. Ist U sogar ein *Normalteiler* von G, d.h. $\forall g \in G : gU = Ug$, dann bildet G/U sogar vermöge $gU * hU := (g * h)U$ eine Gruppe. In jedem Fall aber operiert aber G vermöge

$$(g, H) \mapsto gH := \{gh \mid h \in H\}$$

auf G/U.

Betrachtet man zu $m \in M$ die Stabilisatoruntergruppe G_m, so gibt es den im folgenden Satz angegebenen wichtigen Zusammenhang zwischen den Linksnebenklassen aus G/G_m und der G-Bahn von m.

Seien M und N zwei G-Menge. Eine Abbildung $\varphi : M \to N$ heißt *G-Morphismus* gdw. $\forall m \in M, g \in G$:

$$\varphi(gm) = g\varphi(m)$$

gilt. Eine Abbildung $\varphi : M \to N$ heißt *G-Isomorphismus*, falls φ ein bijektiver G-Morphismus ist (dann ist φ^{-1} ebenfalls G-Morphismus).

4.10 Satz Ist M eine G-Menge und $m \in M$, dann ist die Abbildung

$$\begin{array}{rccc} \varphi : & G/G_m & \to & Gm \\ & gG_m & \mapsto & gm \end{array}$$

bijektiv und es gilt $\forall g \in G : \forall H \in G/G_m : \varphi(gH) = g\varphi(H)$. $\diamond$

Der Satz besagt, dass die Gruppenoperation durch eine Operation auf den Nebenklassen von G/G_m beschrieben wird: Wegen der Bijektivität folgt für endliche Gruppen

4.11 Satz (Bahnenlängenformel)

$$|Gm| = |G/G_m| = [G : G_m] = |G|/|G_m|,$$

◇

Die Bahnlänge von m ist somit gleich der Anzahl der Nebenklassen des Stabilisators G_m. Wegen $\varphi(gH) = g\varphi(H)$ ist es weiterhin egal, ob man die G-Operation auf G/G_m betrachtet oder auf Gm.

Beweis: (Zu Satz 4.10) Wir zeigen zunächst, dass φ wohldefiniert ist. Dazu reicht es zu zeigen, dass aus $gG_m = hG_m$ bereits $gm = hm$ folgt. Wir wissen, dass

$$gG_m = hG_m \Leftrightarrow \exists y \in G_m : g = hy.$$

Dann gilt

$$g = hy \Rightarrow gm = hym = hm,$$

da $y \in G_m$.

φ ist surjektiv, denn für $g \in G$ mit $gm \in Gm$ ist gG_m Linksnebenklasse aus G/G_m und $\varphi(gG_m) = gm$.

φ ist injektiv, denn $gm = hm \Leftrightarrow m = g^{-1}hm$, also $g^{-1}h =: y \in G_m$. Somit folgt $h = gy$, also $hG_m = gG_m$.

Weiterhin $\exists h \in G : H = hG_m$. Dann gilt $\varphi(gH) = \varphi(g(hG_m)) = \varphi((gh)G_m) = (gh)m = g(hm) = g\varphi(hG_m) = g\varphi(H)$. □

Der Hintergrund der im folgenden vorgestellten Indexierungstechnik ist, dass alle Indexobjekte aus M einer G-Bahn zugeordnet werden können. Auf die G-Bahnen wird bezüglich eines Repräsentantensystems $R \subseteq M$ zugegriffen. Hierzu müssen wir zunächst verstehen, wie man von einem $m \in M$ zum zugehörigen Repräsentanten $r = r_m \in R$ kommen kann. Das folgende Resultat besagt, dass dies mit Hilfe der G-Operation bis auf Modifikationen mit dem Stabilisator von r eindeutig möglich ist.

4.12 Satz Sei M eine G-Menge und $r \in M$. Ist $m \in Gr$ und gilt etwa $m = ar$ für $a \in G$, dann gelten:

1. $\{g \in G \mid m = gr\} = aG_r$.
2. $G_m = aG_ra^{-1}$, d.h. Stabilisatoren der Elemente einer G-Bahn sind zueinander konjugierte Untergruppen ($F, H \leq G$ heißen *konjugiert*, falls es ein $g \in G$ gibt, so dass $F = gHg^{-1}$).

◇

Beweis:

1. $m = gr \Leftrightarrow ar = gr \Leftrightarrow a^{-1}g \in G_r \Leftrightarrow g \in aG_r$.
2. Nach 1. gilt

$$aG_r = \{g \in G \mid m = gr\},$$

also äquivalent

$$\begin{aligned} aG_ra^{-1} &= \{g \in G \mid m = gr\}a^{-1} \\ &= \{ga^{-1} \mid m = gr\} \\ &= \{ga^{-1} \mid m = ga^{-1}ar\} \\ &= \{h \in G \mid m = har\} \\ &= \{h \in G \mid m = hm\} = G_m, \end{aligned}$$

da $m = ar$ nach Voraussetzung.

□

Wir ziehen noch eine wichtige Folgerung für den Fall trivialer Stabilisatoren:

4.13 Korollar Gilt unter den Voraussetzungen des vorherigen Satzes $G_r = \{1_G\}$, so besitzt die Gleichung $m = gr$ eine eindeutig bestimmte Lösung in G. ◇

Eine Aussage über die Längen der G-Bahnen wurde in der obigen Bahnenlängenformel getroffen. Eine Aussage über die Anzahl der G-Bahnen trifft folgendes Resultat:

4.14 Satz Die endliche Gruppe G operiere auf der endlichen Menge M. Dann ist die Anzahl der G-Bahnen von M gleich der mittleren Fixpunktanzahl:

$$|\{Gm \mid m \in M\}| = \frac{1}{|G|} \sum_{g \in G} |M_g|,$$

wobei $M_g := \{m \in M \mid gm = m\}$. ◇

Beweis: Übung!

4.2 Konstellationssuche mit $(G, \mathcal{D})$-invertierten Listen

Allgemein können Multimediadokumente wie z.B. Audiosignale, 2D-Bilder, 3D-Datensätze aus der Medizin, Videosignale oder DNA-Sequenzen als Funktionen $f : X \to Y$ beschrieben werden. Bei einem Audiosignal f beschreiben $X \subseteq \mathbb{R}$ die Zeitachse und $Y \subseteq \mathbb{R}$ alle möglichen Amplitudenwerte. f ordnet jedem Zeitpunkt $t \in X$ einen eindeutigen Amplitudenwert $f(t)$ zu. In Anwendungen sind sowohl X als auch Y endlich, d.h., der Graph

$$\text{Graph}(f) := \{(x, f(x)) \mid x \in X\}$$

von f ist eine endliche Teilmenge von $\mathbb{R}^2$. Im folgenden werden wir anstatt mit (der Funktion) f mit dem Graphen von f arbeiten. Ein Vorteil dabei ist, dass in dieser Sichtweise der Definitionsbereich X explizit gemacht wird.
Ein 2D-Bild I kann in ähnlicher Weise als endliche Teilmenge von $\mathbb{R}^2 \times C$ definiert werden, wobei $[(x, y), c] \in I$ aus einer Koordinatenangabe $(x, y) \in \mathbb{R}^2$ und einer Farbinformation $c \in C$ besteht. DNA-Sequenzen kann man als Zeichenketten über dem Alphabet $Y := \{A, C, G, T\}$ modellieren. Die Zeichenkette $Y = (f_1, \ldots, f_n) \in Y^n$ ist dann die Kurzschreibweise der Abbildung $f : X \to Y$ mit $X := [1 : n]$ und $\forall i : f(i) := f_i$. Indem wir auf dem Graphen von f arbeiten, betrachten wir die Menge

$$\{(1, f_1), \ldots, (n, f_n)\}.$$

Wir wollen im folgenden nur *homogene* Datenkollektionen betrachten, d.h. fordern, dass alle Dokumente einer Kollektion vom selben Typ sind. Ausgehend von solch einer homogenen Datenkollektion wählt man zunächst eine geeignete Menge M elementarer Datenobjekte. Im Falle von Audiosignalen könnte dies z.B. die Menge $[t, a]$ aller Zeit-Amplituden Paare sein, also $M = \mathbb{R}^2$. Im Falle von 2D-Bildern könnte man $M = \mathbb{R}^2 \times C$ wählen und $M = \mathbb{N} \times \{A, C, G, T\}$ im Falle von DNA-Sequenzen. Jedes *Dokument* D ist dann eine endliche nichtleere Teilmenge von M, d.h.

$$D \in \mathcal{P}_f(M) := \{X \subseteq M \mid \emptyset \neq X \text{ endlich}\}.$$

Eine Anfrage ist dann ebenfalls eine Teilmenge $Q \subset M$, so dass in diesem Modell kein Unterschied zwischen Dokumenten und Anfragen bestehen (!) Ohne dies formal zu definieren spricht man in der Literatur oft auch von **Query-by-example**. Eine Datenbasis besteht dann aus einer Folge $\mathcal{D} = (D_1, \ldots, D_N)$ von Dokumenten über M.
Wir spezifizieren nun einen Trefferbegriff bezüglich der Operation einer Gruppe G auf der Menge M der Indexobjekte.

4.15 Definition Zu einer Datenbasis $\mathcal{D} = (D_1, \ldots, D_N)$ über einer G-Menge M und einer Anfrage $Q \subseteq M$ ist

$$G_{\mathcal{D}}(Q) := \{(g, i) \mid g \in G, gQ \subseteq D_i\}$$

die *Menge der* $(G, \mathcal{D})$*-Treffer.* ◇

Zur schnellen Trefferbestimmung bemerken wir zunächst, dass mit $G_{\mathcal{D}}(m) := G_{\mathcal{D}}(\{m\})$ für $m \in M$,

$$G_{\mathcal{D}}(Q) = \bigcap_{q \in Q} G_{\mathcal{D}}(q)$$

gilt. Der Beweis geht anlaog zum Beweis des entsprechenden Spezialfalls für die Notensuche. Somit reicht es auch im allgemeinen Fall aus, $(G, \mathcal{D})$-Listen der Form $G_{\mathcal{D}}(m)$ für $m \in M$ zur Trefferbestimmung zu verwenden. Da dies jedoch i.a. unendlich viele Listen wären, versuchen wir auch hier, zu jeder G-Bahn von M

mit nur einer solchen Liste auszukommen. Wir wählen dazu für jede G-Bahn Gm einen Repräsentanten $r_m \in Gm$ und legen nur $G_{\mathcal{D}}(r_m)$ im Suchindex ab. Das folgende Resultat zeigt, dass man dann tatsächlich alle anderen G-Listen $G_{\mathcal{D}}(m')$ zu $m' \in Gm$ rekonstruieren kann.

4.16 Satz Für $m \in M$ und $g \in G$ gilt

$$G_{\mathcal{D}}(gm) = G_{\mathcal{D}}(m)g^{-1} := \{(hg^{-1}, i) \mid (h, i) \in G_{\mathcal{D}}(m)\}.$$

◇

Beweis:

$$\begin{aligned}
(f,i) \in G_{\mathcal{D}}(gm) &\Leftrightarrow fgm \in D_i \\
&\Leftrightarrow (fg,i) \in G_{\mathcal{D}}(m) \\
&\Leftrightarrow \exists (h,i) \in G_{\mathcal{D}}(m) : h = fg \\
&\Leftrightarrow \exists (h,i) \in G_{\mathcal{D}}(m) : f = hg^{-1} \\
&\Leftrightarrow \exists (hg^{-1},i) \in G_{\mathcal{D}}(m)g^{-1} : f = hg^{-1} \\
&\Leftrightarrow (f,i) \in G_{\mathcal{D}}(m)g^{-1}.
\end{aligned}$$

□

4.17 Beispiel $G := (\mathbb{Z} \times \mathbb{Z}, +)$ operiert per Addition in den ersten beiden Komponenten auf der Notenmenge $M := \mathbb{Z} \times \mathbb{Z} \times \mathbb{R}_{>0}$ mit Komponenten für Einsatzzeit, Tonhöhe und Tondauer. Seien

$$D_1 := \{[t_1, p_1, d_1], \ldots, [t_n, p_n, d_n]\}$$

ein Dokument und $m := [t, p, d] \in M$ und $\mathcal{D} := (D_1)$, dann ist

$$G_{\mathcal{D}}(m) = \{(t_i - t, p_i - p, 1) \mid [t_i, p_i, d] \in D_1\}.$$

Dies gilt, denn zu $(t_i - t, p_i - p, 1) \in G_{\mathcal{D}}(m)$ gilt ja äquivalent

$$(t_i - 1, p_i - p)[t, p, d] = (t_i, p_i, d) = (t_i, p_i, d_i) \in D_i.$$

◇

Sei nun $R \subseteq M$ ein Repräsentantensystem bestehend aus genau einem Element pro G-Bahn. Dann gilt:

4.18 Satz Sei $Q \subseteq M$. Dann kann man $q \in Q$ schreiben als $q = g_q r_q$ mit eindeutig bestimmtem $r_q \in R$ und geeignetem $g_q \in G$. Es gilt dann

$$G_{\mathcal{D}}(Q) = \bigcap_{q \in Q} G_{\mathcal{D}}(r_q)g_q^{-1}.$$

◇

Beweis: Es gilt wie oben gesehen $G_{\mathcal{D}}(Q) = \bigcap_{q \in Q} G_{\mathcal{D}}(q)$. Weiterhin folgt aus dem letzten Resultat insbesondere $G_{\mathcal{D}}(q) = G_{\mathcal{D}}(g_q r_q) = G_{\mathcal{D}}(r_q) g_q^{-1}$ für eine beliebige Zerlegung $q = g_q r_q$. Da M disjunkt in G-Bahnen partitioniert ist, und wir ein Repräsentatensystem R der G-Bahnen gewählt haben, ist r_q zu $q \in M$ eindeutig bestimmt. Da G auf den G-Bahnen transitiv operiert, kann man weiterhin mindestens ein g_q mit der geforderten Eigenschaft wählen. □

Einige Bemerkungen insbesondere zur Wahl der g_q:

4.19 Bemerkung

- Mit den Schreibweisen des letzten Satzes erfüllen alle $g \in G$ mit $g r_q = q$ wie in Satz 4.12 gesehen $g \in g_q G_r$.
- Ist der Stabilisator für alle $m \in M$ trivial, so ist die Zerlegung $g_q r_q = q$ im letzten Satz stets eindeutig bestimmt.
- Gilt $m = ar$ für $a \in G$ und sind $g, h \in aG_r$, d.h., $m = gr = hr$, dann gilt auch
$$G_{\mathcal{D}}(r) g^{-1} = G_{\mathcal{D}}(r) h^{-1},$$
d.h. die aus $G_{\mathcal{D}}(r)$ mit Hilfe einer Zerlegung $m = br$ konstruierte G-Liste $G_{\mathcal{D}}(m)$ ist nicht von der Wahl von $b \in aG_r$ abhängig.
- Ist $(g, i) \in G_{\mathcal{D}}(r)$, so gilt auch $gG_r \times \{i\} \subseteq G_{\mathcal{D}}(r)$, denn
$$\begin{aligned} (g,i) \in G_{\mathcal{D}}(r) &\Leftrightarrow gr \in D_i \\ &\Leftrightarrow \forall h \in G_r : ghr \in D_i \\ &\Leftrightarrow gG_r r \in D_i \\ &\Leftrightarrow gG_r \times \{i\} \subseteq G_{\mathcal{D}}(r). \end{aligned}$$
Je nach Anwendung kann es sinnvoll sein, anstatt der Nebenklasse gG_r von Gruppenelementen nur einen Repräsentanten der Nebenklasse zur Speicherung im Index zu verwenden. Für
$$(G/G_r)_{\mathcal{D}}(m) := \{(g, i) \mid g = hG_r \wedge hG_r m \subseteq D_i\}$$
läuft dies auf eine kompakte Speicherung aller g's mit $(g, i) \in (G/G_r)_{\mathcal{D}}(m)$ hinaus. Bei der Trefferbestimmung muß dies dann wieder expandiert werden, denn
$$G_{\mathcal{D}}(m) = \bigcup_{(g,i) \in (G/G_r)_{\mathcal{D}}(m)} g \times \{i\}.$$

◇

Zusammenfassend wählen wir zu einer Gruppenoperation von G auf M ein Repräsentantensystem $R \subseteq M$ mit genau einem Element pro G-Bahn und erstellen den Suchindex als
$$(G_{\mathcal{D}}(r))_{r \in R}.$$

4.3 Fehlertoleranz

Die Konzepte zur fehlertoleranten Suche, die wir im Falle der Notensuche kennengelernt haben, lassen sich auf das allgemeine, gruppentheoretisch erklärte Suchszenario übertragen.

4.3.1 Toleranz gegenüber Fehlstellen

Die gegenüber maximal k Fehlstellen tolerante Suche läßt sich analog zum Fall der Notensuche auch für den allgemeinen Fall erreichen. Für eine Anfrage $Q \subseteq M$ und $k \in \mathbb{N}$ sei

$$G_{\mathcal{D},k}(Q) := \{(g,i) \mid |gQ \setminus D_i| \leq k\}$$

die Menge aller *Treffer mit höchstens k Fehlstellen* (k-Mismatch-Treffer). Die Berechnung von $G_{\mathcal{D},k}(Q)$ kann wieder effizient mit einem auf dynamischer Programmierung basierenden Algorithmus geschehen.
Hierzu sei $Q =: \{q_1, \ldots, q_n\}$ eine beliebige Numerierung der Elemente aus Q. Definiere $G_j := G_{\mathcal{D}}(q_j)$ als Liste zum j-ten Element. Weiterhin definieren wir die Mengen der Trefferkandidaten im j-ten Schritt des Algorithmus als $\Gamma^1 := G_1$ und für $j \geq 2$

$$\Gamma^j := \{\gamma \in (\Gamma^{j-1} \cup G_j) \mid C^j(\gamma) > 0\}.$$

Hierbei verwenden wir wieder Kreditfunktionen $C^j : \Gamma^{j-1} \cup G_j \to \mathbb{Z}$ mit $C^1(\gamma) := k+1$ und für $j \geq 2$

$$C^j(\gamma) := \begin{cases} C^{j-1}(\gamma) & \text{falls } \gamma \in \Gamma^{j-1} \cap G_j, \\ C^{j-1}(\gamma) - 1 & \text{falls } \gamma \in \Gamma^{j-1} \setminus G_j, \\ k+2-j & \text{falls } \gamma \in G_j \setminus \Gamma^{j-1}. \end{cases}$$

Offensichtlich gilt:

4.20 Satz $G_{\mathcal{D},k}(Q) = \Gamma^n$. ◇

4.3.2 Fuzzy-Anfragen

Das Konzept der Fuzzy-Anfragen läßt sich ebenfalls problemlos auf den allgemeinen Fall übertragen. Sei $(F_1, \ldots, F_n)$ eine Folge von Alternativenmengen (oder *Fuzzymengen*) $F_i \subset M$. Eine *Fuzzy-Anfrage* ist eine Familie von Anfragen

$$\mathcal{Q} := \{\{q_1, \ldots, q_n\} \mid \forall i : q_i \in F_i\}.$$

Die Menge aller *Fuzzy-Treffer* ist definiert als

$$G_{\mathcal{D}}(\mathcal{Q}) := \{(g,i) \mid \exists Q \in \mathcal{Q} : gQ \subseteq D_i\} = \bigcup_{Q \in \mathcal{Q}} G_{\mathcal{D}}(Q).$$

Die Fuzzy-Treffer können schnell bestimmt werden, da

$$G_{\mathcal{D}}(\mathcal{Q}) = \bigcap_{i=1}^{n} \left(\bigcup_{q \in F_i} G_{\mathcal{D}}(q) \right).$$

gilt. (Übung!)

4.4 Einbeziehung von Vorwissen

Im folgenden Abschnitt wollen wir untersuchen, wie etwaiges Vorwissen über die gesuchten Treffer zur Steigerung der Effizient eingesetzt werden kann. Wir betrachten dazu zunächst das Beispiel der Musiksuche.
Angenommen, ein Benutzer weiß, dass seine Anfrage ein Stück im 4/4-tel Takt betrifft. Sein musikalisches Gefühl sagt ihm, dass seine Anfrage mit einem Auftakt beginnt. Wenn er nun weiß, dass jeder 4/4-tel Takt in Sechzehnteln quantisiert wurde, so hat er hinsichtlich der metrischen Position seiner Anfrage ein Vorwissen: er kann die Einsatzzeiten der Noten bis auf die absolute Taktnummer genau angeben. Statt $G = \mathbb{Z} \times \mathbb{Z}$ braucht er aufgrund seines Vorwissens nur noch die Untergruppe $U = 16\mathbb{Z} \times \mathbb{Z}$. Besitzt der Benutzer zudem ein absolutes Gehör, so wird er bei seiner Anfrage auch die Tonhöhe exakt angeben können. Statt U kommt er dann mit der wesentlich kleineren Untergruppe $V = 16\mathbb{Z} \times \{0\}$ aus. Im allgemeinen gibt es, wenn man nur die 128 MIDI-Tonhöhen berücksichtigt und eine Sechzehntel-Quantisierung vornimmt, genau $128 \cdot 16 = 2048$ V-invertierte Listen.
Allgemein wird das Vorwissen der Benutzer beschrieben durch Untergruppen von G. Je größer das Vorwissen, desto kleiner ist die Untergruppe. Den Effizienzgewinn wollen wir nun genauer untersuchen. Ist also U eine Untergruppe von G, so kann man analog zu den G-invertierten Listen $G_{\mathcal{D}}(m)$ auch U-invertierte Listen $U_{\mathcal{D}}(m) := \{(u, i) \mid u \in U, um \in D_i\}$ bilden. Typischerweise sind U-invertierte Listen kürzer als G-invertierte Listen. G- und U-invertierte Listen hängen wie folgt zusammen:

4.21 Satz Ist $G = \sqcup_{h \in H} Uh$ eine Zerlegung von G in *Rechts*nebenklassen nach der Untergruppe U, so gilt für alle $m \in M$:

$$G_{\mathcal{D}}(m) = \bigsqcup_{h \in H} U_{\mathcal{D}}(hm)h.$$

◇

Beweis: Für $g \in G$, $h \in H$ und $u \in U$ mit $g = uh$ gilt:

$$\begin{aligned} (g, i) \in G_{\mathcal{D}}(m) &\Leftrightarrow (uh)m \in D_i \Leftrightarrow u(hm) \in D_i \Leftrightarrow (u, i) \in U_{\mathcal{D}}(hm) \\ &\Leftrightarrow (g, i) \in U_{\mathcal{D}}(hm)h. \end{aligned}$$

□

Den durch Berücksichtigung des Zusatzwissens erzielten Gewinn bei der Bestimmung der Treffermenge beschreibt folgender

4.22 Satz Mit den Bezeichnungen des letzten Satzes gilt für eine Anfrage Q:

$$G_{\mathcal{D}}(Q) = \bigsqcup_{h \in H} U_{\mathcal{D}}(hQ)h.$$

◇

Beweis: Da für $h \in H$ und $m \in M$ die erste Projektion der Menge $U_{\mathcal{D}}(hm)h$ Teilmenge der Rechtsnebenklasse Uh ist und diese Nebenklassen zu verschiedenen Elementen aus H disjunkt sind, gilt zusammen mit Satz 4.21

$$\begin{aligned} G_{\mathcal{D}}(Q) &= \bigcap_{q \in Q} G_{\mathcal{D}}(q) = \bigcap_{q \in Q} \left(\bigsqcup_{h \in H} U_{\mathcal{D}}(hq)h \right) \\ &= \bigsqcup_{h \in H} \left(\bigcap_{q \in Q} U_{\mathcal{D}}(hq) \right) h = \bigsqcup_{h \in H} U_{\mathcal{D}}(hQ)h. \end{aligned}$$

□

Arbeiten wir mit U- statt mit G-invertierten Listen, so zeigt der letzte Satz das Einsparungspotential: die Liste $G_{\mathcal{D}}(Q)$ setzt sich aus vielen Listen der Form $U_{\mathcal{D}}(hQ)h$ zusammen. Zur Problemlösung brauchen wir aber aufgrund des Vorwissens nur die Liste zu $h = 1$, also $U_{\mathcal{D}}(Q)$.

4.5 Literatur

Für eine ausführlichere Darstellung der gruppenbasierten Konstellationssuche verweisen wir wieder auf die Habilitationsschrift von F. Kurth [21]. Die verallgemeinerte Suchtechnik wurde von M. Clausen, F. Kurth und H. Körner entwickelt. Eine Originalarbeit ist hier [22]. Eine Einführung mit Tutorialcharakter wurde im Rahmen eines zweiwöchigen Advanced Study Institutes der NATO vorgestellt [23]. Die Anwendung der Technik auf das 3D-Objektretrieval untersucht A. Mosig in seiner Diplomarbeit [24], eine Anwendung auf die 2D-Bildsuche ist Gegenstand der Diplomarbeit von T. Röder [25]

Literaturverzeichnis

[1] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3), 1980.

[2] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[3] William B. Frakes and Ricardo A. Baeza-Yates, editors. *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992.

[4] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes*. Van Nostrand Reinhold, 2nd edition, 1999.

[5] Maxime Crochemore and Wojcieich Rytter. *Text Algorithms*. Oxford Univ. Press, 1994. CRO m 94:1 P-Ex.

[6] Hans-Joachim Böckenhauer and Dirk Bongartz. *Algorithmische Grundlagen der Bioinformatik*. Teubner, 1. aufl. edition, 2003.

[7] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[8] Ulrich Krengel. *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Vieweg, 1991.

[9] Robert Schaback and Helmut Werner. *Numerische Mathematik*. Springer, 4 edition, 1993.

[10] Eugene Seneta. *Non–Negative Matrices*. George Allen & Unwin Ltd., 1973.

[11] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[12] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–??, 1999.

[13] Jeffrey Dean and Monika R. Henzinger. Finding related pages in the World Wide Web. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1467–1479, 1999.

[14] Josef Stoer. *Numerische Mathematik 1.* Springer, 6th edition, 1993.

[15] Josef Stoer and Roland Bulirsch. *Numerische Mathematik 2.* Springer, 3rd edition, 1990.

[16] Eleanor Selfridge-Field, editor. *Beyond MIDI: The Handbook of Musical Codes.* MIT Press, 1997.

[17] Michael Clausen, Roland Engelbrecht, Dirk Meyer, and Jürgen Schmitz. PROMS: A Web-based Tool for Searching in Polyphonic Music. In *Proceedings Intl. Symp. on Music Information Retrieval 2000, Plymouth, M.A., USA*, 2000.

[18] Michael Clausen and Frank Kurth. A Unified Approach to Content-Based and Fault Tolerant Music Identification. In *International Conference on Web Delivery of Music, Darmstadt, Germany*, 2002.

[19] Michael Clausen and Frank Kurth. A Unified Approach to Content-Based and Fault Tolerant Music Recognition. *IEEE Transactions on Multimedia*, 6(5), October 2004.

[20] Frank Kurth, Michael Clausen, and Andreas Ribbrock. Efficient Fault Tolerant Search Techniques for Full-Text Audio Retrieval. In *Proc. 112th AES Convention, Munich, Germany*, 2002.

[21] Frank Kurth. Beiträge zum effizienten Multimediaretrieval. Habilitationsschrift, Mathematisch-Naturwissenschaftliche Fakultät der Universität Bonn, 2004.

[22] Michael Clausen, Heiko Körner, and Frank Kurth. An Efficient Indexing and Search Technique for Multimedia Databases. In *SIGIR Workshop on Multimedia Retrieval, Toronto, Canada*, 2003.

[23] Michael Clausen and Frank Kurth. Content-Based Information Retrieval by Group Theoretical Methods. In *NATO Advanced Study Institute (ASI), Lucca, Italy*, 2003.

[24] Axel Mosig. Algorithmen und Datenstrukturen zur effizienten Konstellationssuche. Diplomarbeit, Institut für Informatik III, Universität Bonn, 2001.

[25] Tido Röder. A Group Theoretical Approach to Content-Based Image Retrieval. Diplomarbeit, Institut für Informatik III, Universität Bonn, 2002.