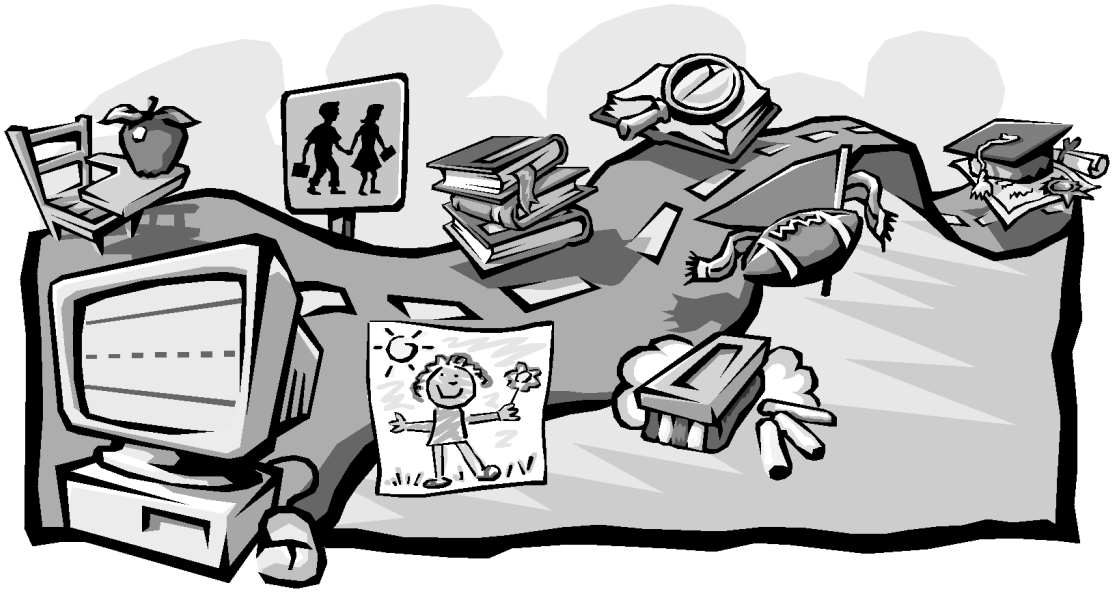
Information Retrieval

Grundlagen, Modelle, Implementierung und Anwendungen

Andreas Henrich

Praktische Informatik, Fakultät Sozial- und Wirtschaftswissenschaften
Otto-Friedrich Universität Bamberg, 96045 Bamberg
Email: andreas.henrich@sowi.uni-bamberg.de



Inhaltsverzeichnis

Kapitel 1

Motivation

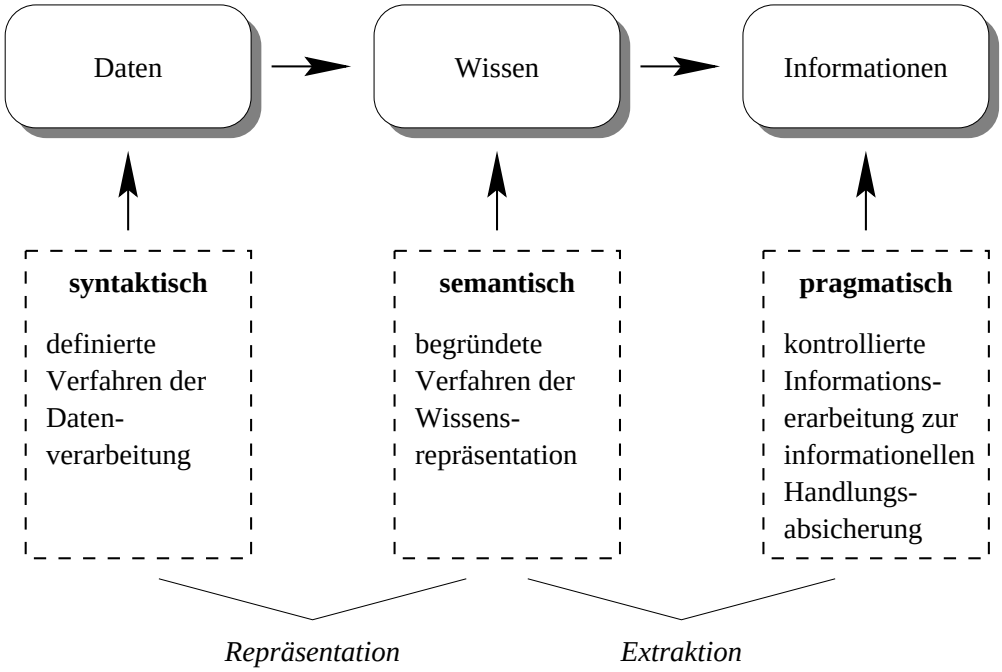
- Was will Information Retrieval?
- Worin unterscheidet sich Information Retrieval von Datenbanken?
- Was werden wir in der Vorlesung behandeln?

1.1 Der Begriff Information Retrieval

Information Retrieval setzt sich als Begriff aus zwei Teilen zusammen:

- *Information*
  Daten (*syntaktisch*) — Wissen (*semantisch*) — Information (*pragmatisch*)
  „Information ist die Teilmenge von Wissen, die von jemandem in einer konkreten Situation zur Lösung von Problemen benötigt wird“ [Kuh90]
- *Retrieval*
  Einige Bedeutungen von *retrieve* aus Langenscheidts Wörterbuch [Lan87]:
  - wiederfinden, wiederbekommen
  - (sich et.) zurückholen
  - herausholen, herausfischen
  - wiedergewinnen, wiedererlangen
  - et. der Vergangenheit entreißen
  - apportieren

Begriffsverständnis der deutschen Informationswissenschaft (nach [Fuh96])



Aufgabe eines IR-Systems: Transformation von Daten/Wissen in Information!

Die Ausgangssituation beim Information Retrieval ist damit:

- Jemand steckt in einer konkreten Situation.
- In dieser Situation sucht er für ihn interessante „Dokumente / Objekte“.
- Ein Information Retrieval System soll ihn dabei *unterstützen*

Es geht also um eine

„Inhaltsbasierte Suche nach Dokumenten / Objekten“

Offene Fragen:

- Was bedeutet „inhaltsbezogen“?
- Welche Arten von Dokumenten / Objekten werden betrachtet?

Die historische Sicht:

- Suche nach Büchern (Dokumenten) in Bibliotheken
- Probleme:
  1. Die Anordnung der Bücher in den Regalen
     (ein- bis maximal zwei-dimensional!)
  2. Das Führen von Katalogen
     - nach Autor
     - nach Titelstichwörtern, ...
- Lösungsansätze:
  - Klassifikationsschemata
  - manuelle Verschlagwortung
  - manuelle Erfassung bibliographischer Informationen

Neuere Entwicklungen

- Internet
  - bibliographische Informationen nur teilweise verfügbar

  ⇒ stärkeres Gewicht auf den Inhalt

  - zunehmende Strukturierung der Dokumente (XML)

  ⇒ Ausnutzung der Struktur bei der Suche

  - extrem schnelles Wachstum der Informationsmenge

  ⇒ manuelle Indexierung nicht möglich

- Multimedia
  - neue Medien (Grafik, Audio, Video, Animation, ...)

  ⇒ Suche auch nach den Inhalten dieser Medien

  - Kombination mehrerer Medien

  ⇒ Berücksichtigung der Kombination
    - z.B. Suche eines Bildes über die Bildunterschrift
    - z.B. inhaltliche Betrachtung von Bild und Text

1.2 Ansätze zum Information Retrieval

1. Klassifikation
   - Dokumente werden – i.allg. manuell – in ein Klassifikationsschema eingeordnet.
   - Die Zuordnung muß i.allg. eindeutig sein.
   - Ziel ist das physisch benachbarte Aufstellen der Dokumente z.B. in einer Bibliothek.
2. Anfrage
   - Der „Benutzer“ formuliert seinen „Informationswunsch“ als Anfrage.
   - Das System versucht (automatisch) hierzu relevante Dokumente zu ermitteln.
3. Browsing
   - Der Benutzer will die „Dokumentenkollektion“ interaktiv erarbeiten.
   - Das System sollte ihm dazu eine inhaltliche Clusterung der Dokumente anbieten.
   - Von einem Dokument komme ich dann einfach zu „inhaltlich ähnlichen“.
4. Information Filtering
   - Aus einem andauernden Strom von Dokumenten (Beispiel Nachrichten-Ticker) sollen automatisch die den Benutzer interessierenden ermittelt werden.

1.3 Abgrenzung von Information Retrieval und Fakten Retrieval

	Fakten Retrieval	Information Retrieval
Matching	Exact match	Partial match, best match
Inference	Deduction	Induction
Model	Deterministic	Probabilistic
Classification	Monothetic	Polythetic
Query language	Artificial	Natural
Query specification	Complete	Incomplete
Items wanted	Matching	Relevant
Error response	Sensitive	Insensitive

Die nicht hervorgehobenen Bereiche werden im weiteren noch einzeln betrachtet.

Matching

- *Fakten Retrieval: Exact match*
  Die Zugehörigkeit zum Ergebnis ist eindeutig entscheidbar.
- *Information Retrieval: Partial match, best match*
  Es gibt einen sanften Übergang und keine harte Grenze.

Situationen, in denen *partial match* sinnvoll ist, findet man auch in typischen „Datenbankanwendungen“:

Beispiel: Auswahl in einer Datenbank

Das Problem:

In einer Datenbank mit Gebrauchtwagen
soll ein passendes Exemplar gesucht werden!

Gebrauchtwagen

G-NR	MODELL	KLIMA	BAUJAHR	KW	KILOMETER	PREIS
132	601	ja	1994	45	19000	27000
472	601	ja	1993	80	10000	17000
173	601	ja	1995	45	23000	19000
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Gesucht wird: ein 601

- mit weniger als 20000 km
- nach Baujahr 1993
- mit mindestens 45 KW
- möglichst günstig

die Anfrage in SQL:

```
SELECT G-NR
FROM Gebrauchtwagen
WHERE KILOMETER < 20000
  AND BAUJAHR > 1993
  AND KW >= 45
ORDER PREIS ASC
```

Problem: die Bedingungen sind eigentlich vage!

Inference

- *Fakten Retrieval: Deduction*

  Herleitung des Besonderen aus dem Allgemeinen.

  In einer Anfrage werden die gesuchten Datensätze (das Besondere) aus der Datenbasis (dem Allgemeinen) extrahiert.

- *Information Retrieval: Induction*

  Herleitung von allgemeinen Regeln aus Einzelfällen.

  Natürlich enthält auch IR dekuktive Aspekte, aber zusätzlich gibt es induktive Aspekte.

  Beispiel: Aus einem von dem Benutzer für relevant erklärten Dokument werden Rückschlüsse auf seinen Informationswunsch gezogen.

Model

- *Fakten Retrieval: Deterministic*

  Da es keine „Vagheit“ gibt, ist das Schließen deterministisch.

- *Information Retrieval: Probabilistic*

  Es gibt im IR zahlreiche mögliche Quellen der Unschärfe:

  - die Formulierung der Anfrage z.B. in natürlicher Sprache
  - die Unschärfe der verwalteten Dokumente
  - das induktive Schließen

Classification

- *Fakten Retrieval: Monothetic*

  Wenn man die Einteilung der Objekte in Klassen betrachtet, dann sind die Attributwerte der Objekte, die ja immer alle vorhanden sind, notwendig und hinreichend zur Einteilung.

- *Information Retrieval: Polythetic*

  Da beim IR i.allg. kein festes Schema existiert, sind u.U. zu den Dokumenten unterschiedliche Informatioen vorhanden.

  Folge: eine eindeutige Zuordnung ist nicht möglich.

Error response

- *Fakten Retrieval: Sensitive*

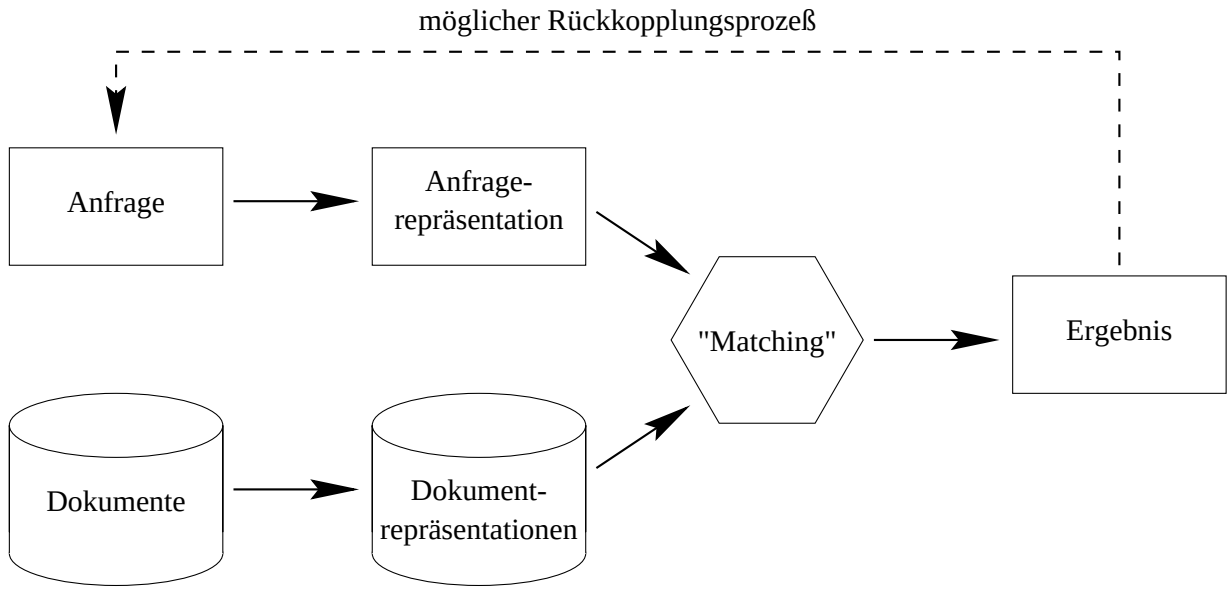
  Fehler im Ergebnis sind nicht tolerierbar.

- *Information Retrieval: Insensitive*

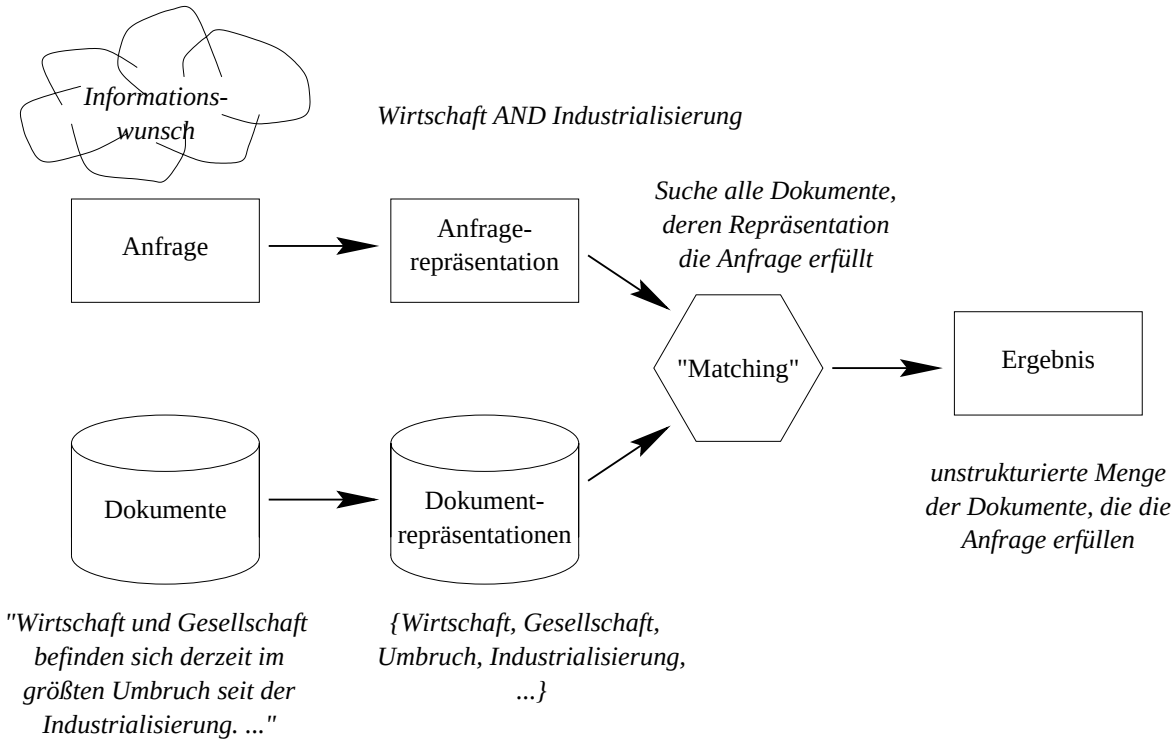
  Da das Ergebnis einer Anfrage ohnehin nicht als korrekt oder nicht korrekt eingestuft werden kann, ist es u.U. vertretbar leichte Fehler zuzulassen.

  So vereinfachen sich z.B. einige Verfahren zum IR deutlich wenn man einige Zwischenrechnungen nur grob durchführt.

1.4 Das grundsätzliche Vorgehen beim Information Retrieval



Beispiel: boolesches Retrieval



Boolesches Retrieval

Repräsentation der Dokumente:

- Menge der vorkommenden Terme

Formulierung des Informationswunsches:

- boolescher Ausdruck über die Terme, die im Dokument vorkommen sollen/dürfen

Text:

„Die Regelstudienzeit beträgt einschließlich der Zeit für Abschlußprüfung und Anfertigung der Diplomarbeit acht Semester und drei Monate. Das Studium gliedert sich in ein viersemestriges Grundstudium und ein viersemestriges Hauptstudium. Der Höchstumfang der erforderlichen Lehrveranstaltungen im Pflicht- und Wahlpflichtbereich beträgt 136 Semesterwochenstunden. Das gemäß §29 Abs. 1 Nr. 5 abzuleistende Praktikum wird auf die Regelstudienzeit nicht angerechnet.“

Textrepräsentation:

{ „Abschlußprüfung“, „Anfertigung“, „Diplomarbeit“, „Grundstudium“, „Hauptstudium“, „Höchstumfang “, „Lehrveranstaltungen“, „Monate“, „Pflicht“, „Praktikum“, „Regelstudienzeit“, „Semester“, „Semesterwochenstunden“, „Studium“, „Wahlpflichtbereich“, „Zeit“}

Beispielanfrage:

Studium $\wedge$ Diplomarbeit $\wedge \neg$ Praktikum

Schwächen des booleschen Retrieval:

- keine Rückführung der Wörter auf eine Grundform
- keine Gewichtung der Wörter
  - nach dem Ort des Vorkommens
  - nach der Häufigkeit des Vorkommens
- keine Zerlegung von Mehrwortgruppen
- relativ aufwendige Formulierung der Anfrage
- kaum vorhersehbare Ergebnisgröße
- kein Ranking der Dokumente

Mögliche Verbesserungen:

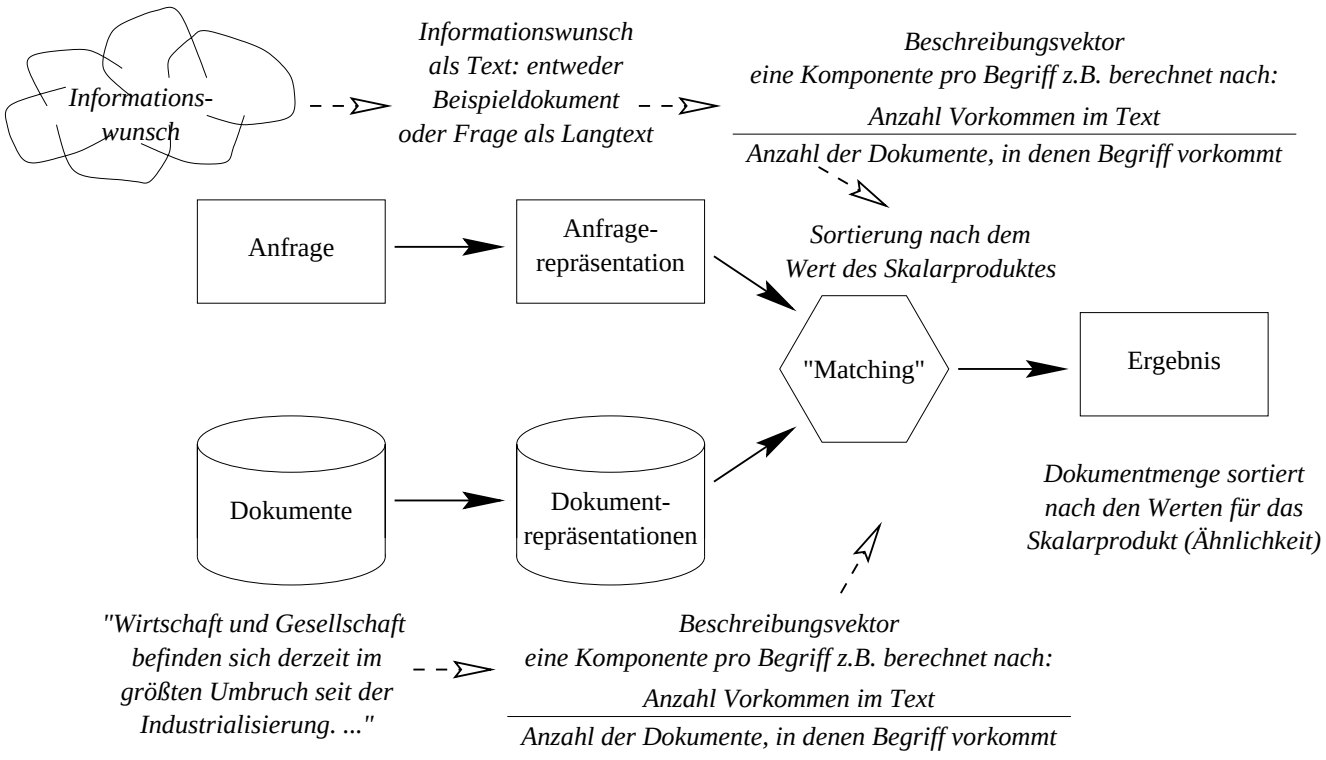
- Rückführung aller Wörter auf ihre Grundform
- Berücksichtigung des Vorkommensortes eines Wortes
  - im Titel
  - in einer Überschrift
  - in der Zusammenfassung
- Berücksichtigung der Unterscheidungskraft des Wortes
  - Wort kommt in vielen Dokumenten vor
    $\Rightarrow$ Unterscheidungskraft niedrig
  - Wort kommt in wenigen Dokumenten vor
    $\Rightarrow$ Unterscheidungskraft hoch

- Berücksichtigung weiterer, die Relevanz potentiell beeinflussender Faktoren
  - Wann wurde das Dokument geschrieben?
  - Wer hat das Dokument erstellt?

⇒ Berechnung einer *Kennzahl* für jedes Dokument
  - die eine Abschätzung der Relevanzwahrscheinlichkeit ist
  - die ein Ranking der Dokumente erlaubt

- Iterative Verbesserung des Ergebnisses
  - durch eine nutzerseitige Beurteilung der Dokumente im vorliegenden Ergebnis

Beispiel: das Vektorraummodell (ganz grob)



Das Vektorraummodell nach Salton (etwas genauer)

Vergleich von Beschreibungsvektoren für Dokumente und Anfragetext.

Ausgangspunkt: **Vokabular** mit t Begriffen (die, die in den Dokumenten vorkommen)

Wir brauchen nun einige Definitionen:

N = Anzahl der Dokumente in der Dokumentenkollektion
n_k = Anzahl der Dokumente, die den Begriff/Term k enthalten
tf_{dk} = Vorkommenshäufigkeit von Begriff k in Dokument D

Dokument D repräsentiert durch Vektor: $\mathcal{D} = (w_{d1}, w_{d2}, \ldots, w_{dt})$

w_{dk} entspricht der Relevanz von Dok D für den Begriff k:

$$w_{dk} = \frac{tf_{dk} \cdot \log \frac{N}{n_k}}{\sqrt{\sum_{i=1}^{t} \left(tf_{di} \cdot \log \frac{N}{n_i} \right)^2}}$$

Die Anfrage Q wird ebenfalls durch einen Vektor repräsentiert: $\mathcal{Q} = (w_{q1}, w_{q2}, \ldots, w_{qt})$

die Komponenten werden nach folgender Formel berechnet:

$$w_{qk} = \begin{cases} \left(0.5 + \frac{0.5 \cdot tf_{qk}}{\max\limits_{1 \le i \le t} tf_{qi}}\right) \cdot \log \frac{N}{n_k} & \text{if} \quad tf_{qk} > 0 \\ 0 & \text{if} \quad tf_{qk} = 0 \end{cases}$$

Die Ähnlichkeit zwischen einer Anfrage und einem Dokument kann dann durch das Skalarprodukt beschrieben werden:

$$\text{similarity}(\mathcal{Q}, \mathcal{D}) = \sum_{k=1}^{t} w_{qk} \cdot w_{dk}$$

Beispiel zum Vektorraummodell:

- D_1 = „Häuser in Italien“
- D_2 = „Häuser in Italien und um Italien“
- D_3 = „Gärten und Häuser in Italien“
- D_4 = „Gärten in Italien“
- D_5 = „Gärten und Häuser in Frankreich“
- Q = „Häuser in Italien“

Damit besteht das Vokabular aus den Begriffen: *Häuser*, *Italien* , *Gärten*, *Frankreich*

Wir erhalten $t = 4$ und $N = 5$

Wir legen für die Beschreibungsvektoren fest:

Häuser = Komponente 1, *Italien* = Komponente 2, *Gärten* = Komponente 3, *Frankreich* = Komponente 4

Für D_2 erhalten wir somit:

$tf_{2;1} = 1$, $tf_{2;2} = 2$, $tf_{2;3} = 0$, $tf_{2;4} = 0$

Ferner gilt:

$n_1 = 4$, $n_2 = 4$, $n_3 = 3$, $n_4 = 1$

$$\mathcal{D}_2 = \begin{pmatrix} \frac{1 \cdot \log \frac{5}{4}}{\sqrt{\left(1 \cdot \log \frac{5}{4}\right)^2 + \left(2 \cdot \log \frac{5}{4}\right)^2 + \left(0 \cdot \log \frac{5}{3}\right)^2 + \left(0 \cdot \log \frac{5}{1}\right)^2}} \\ \frac{2 \cdot \log \frac{5}{4}}{\sqrt{\left(1 \cdot \log \frac{5}{4}\right)^2 + \left(2 \cdot \log \frac{5}{4}\right)^2 + \left(0 \cdot \log \frac{5}{3}\right)^2 + \left(0 \cdot \log \frac{5}{1}\right)^2}} \\ \frac{0 \cdot \log \frac{5}{3}}{\sqrt{\left(1 \cdot \log \frac{5}{4}\right)^2 + \left(2 \cdot \log \frac{5}{4}\right)^2 + \left(0 \cdot \log \frac{5}{3}\right)^2 + \left(0 \cdot \log \frac{5}{1}\right)^2}} \\ \frac{0 \cdot \log \frac{5}{1}}{\sqrt{\left(1 \cdot \log \frac{5}{4}\right)^2 + \left(2 \cdot \log \frac{5}{4}\right)^2 + \left(0 \cdot \log \frac{5}{3}\right)^2 + \left(0 \cdot \log \frac{5}{1}\right)^2}} \end{pmatrix} = \begin{pmatrix} 0,4472 \\ 0,8944 \\ 0 \\ 0 \end{pmatrix}$$

Insgesamt erhalten wir folgende Beschreibungsvektoren:

$$\mathcal{D}_1 = \begin{pmatrix} 0,7071 \\ 0,7071 \\ 0 \\ 0 \end{pmatrix}, \quad \mathcal{D}_2 = \begin{pmatrix} 0,4472 \\ 0,8944 \\ 0 \\ 0 \end{pmatrix}, \quad \mathcal{D}_3 = \begin{pmatrix} 0,3716 \\ 0,3716 \\ 0,8508 \\ 0 \end{pmatrix}, \quad \mathcal{D}_4 = \begin{pmatrix} 0 \\ 0,4003 \\ 0,9164 \\ 0 \end{pmatrix}, \quad \mathcal{D}_5 = \begin{pmatrix} 0,1310 \\ 0 \\ 0,2999 \\ 0,9449 \end{pmatrix}, \quad \mathcal{Q} = \begin{pmatrix} 0,0969 \\ 0,0969 \\ 0 \\ 0 \end{pmatrix}$$

Es ergibt sich folgendes Ranking:

1. D_1 mit similarity$(\mathcal{Q}, \mathcal{D}_1) = 0,137$;
2. D_2 mit similarity$(\mathcal{Q}, \mathcal{D}_2) = 0,130$;
3. D_3 mit similarity$(\mathcal{Q}, \mathcal{D}_3) = 0,072$;
4. D_4 mit similarity$(\mathcal{Q}, \mathcal{D}_4) = 0,039$;
5. D_5 mit similarity$(\mathcal{Q}, \mathcal{D}_5) = 0,013$;

Probabilistisches Information Retrieval

- Man wähle einige quantifizierbare, die Relevanz eines Dokumentes D in Relation zu einer Anfrage A bestimmende Eigenschaften aus.

  Beispiele:

  – die Anzahl der übereinstimmenden Terme

  – die Anzahl der Terme aus der Anfrage, die im Titel des Dokumentes auftreten

  – die Anzahl der Terme im Dokument

  – die Vorkommenshäufigkeit der 3 „wichtigsten“ gemeinsamen Terme im Dokument

  – ...

- Man fasse die Werte für Dokument D und Anfrage A zu einem Vektor $\vec{V}_{DA}$ zusammen.

- Man definiere eine Skala für die Relevanz eines Dokumentes (z.B. $0 \ldots 10$).
- Man bestimme für eine Menge von Dokumenten und Anfragen manuell die Relevanzwerte $\mathcal{R}$.
- Man bestimme eine Funktion $\mathcal{F}$, die
  - aus der Menge der Beschreibungsvektoren in das Intervall $[0; 10]$ abbildet und
  - für die gegebene Stichprobe die Abweichungsquadrate minimiert.
- Nun kann man $\mathcal{F}$ auf beliebige Kombinationen aus Anfrage und Dokument in der Kollektion anwenden.
- Man erhält so für eine beliebige Anfrage ein Ranking der Dokumente.
- Das Ranking kann durch Weiterentwicklung von $\mathcal{F}$ weiter verbessert werden.

Beispiele für die Formulierung des Informationswunsches

- mehr oder weniger unstrukturierte Angabe von „Suchtermen“
  Beispielanfrage:

  Prozessor Intel Architektur Pentium

- Angabe eines (ggf. erweiterten) booleschen Ausdrucks:

  Bundespräsident NEAR(4) *Wahl* AND *FDP*

- Angabe des Informationswunschs in natürlicher Sprache:

  Gesucht werden alle Zeitungsartikel, die sich mit der Rolle der FDP bei der Wahl des Bundespräsidenten beschäftigen.

- Anfrage an eine Datenbank:

  SELECT * FROM *Artikel* WHERE *Text* contains *„Bundespräsident“*

Darstellung und Beurteilung des Ergebnisses

strukturiertes oder unstrukturiertes Ergebnis

- Beispiel *boolesches Retrieval* oder *Datenbankanfrage*:

  Das Ergebnis der Anfrage ist eine unstrukturierte Menge von Dokumenten

- Strukturierung: Versuch der absteigenden Sortierung nach Relevanz

  Beispiel: Suchanfrage enthält vier Terme:

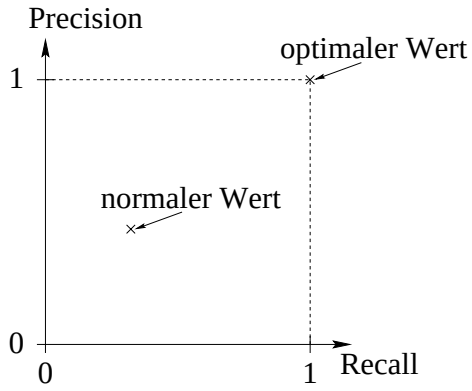
  Dokumente, die alle 4 Terme enthalten werden vor Dokumenten angezeigt, die nur 3 oder weniger der Terme enthalten.

Beurteilung

- Kriterien bei unstrukturierten Ergebnissen:

$$\text{Recall} = \frac{\text{gefundene relevante Dokumente}}{\text{relevante Dokumente insgesamt}}$$

$$\text{Precision} = \frac{\text{gefundene relevante Dokumente}}{\text{gefundene Dokumente insgesamt}}$$



- Kriterien bei unstrukturierten Ergebnissen: ? (näheres später)

„Definition“ zum Information Retrieval von der Fachgruppe IR der GI:

Im Information Retrieval (IR) werden Informationssysteme in bezug auf ihre Rolle im Prozeß des Wissenstransfers vom menschlichen Wissensproduzenten zum Informations-Nachfragenden betrachtet. Die Fachgruppe „Information Retrieval“ in der Gesellschaft für Informatik beschäftigt sich dabei schwerpunktmäßig mit jenen Fragestellungen, die im Zusammenhang mit vagen Anfragen und unsicherem Wissen entstehen. Vage Anfragen sind dadurch gekennzeichnet, daß die Antwort a priori nicht eindeutig definiert ist. Hierzu zählen neben Fragen mit unscharfen Kriterien insbesondere auch solche, die nur im Dialog iterativ durch Reformulierung (in Abhängigkeit von den bisherigen Systemantworten) beantwortet werden können; häufig müssen zudem mehrere Datenbasen zur Beantwortung einer einzelnen Anfrage durchsucht werden. Die Darstellungsform des in einem IR-System gespeicherten Wissens ist im Prinzip nicht beschränkt (z.B. Texte, multimediale Dokumente, Fakten, Regeln, semantische Netze). Die Unsicherheit (oder die Unvollständigkeit) dieses Wissens resultiert meist aus der begrenzten Repräsentation von dessen Semantik (z.B. bei Texten oder multimedialen Dokumenten); darüber hinaus werden auch solche Anwendungen betrachtet, bei denen die gespeicherten Daten selbst unsicher oder unvollständig sind (wie z.B. bei vielen technisch-wissenschaftlichen Datensammlungen). Aus dieser Problematik ergibt sich die Notwendigkeit zur Bewertung der Qualität der Antworten eines Informationssystems, wobei in einem weiteren Sinne die Effektivität des Systems in bezug auf die Unterstützung des Benutzers bei der Lösung seines Anwendungsproblems beurteilt werden sollte.

1.5 Aufbau der Vorlesung

- Evaluierung von IR-Systemen
- Berücksichtigung der Vagheit in Sprache(n)
- Klassifikationsansätze
- Pattern Matching
- Einfache IR Modelle und ihre Implementierung
- Das Vektorraummodell
- Probabilistisches IR
- Dokumentenklusterung
- Formate zur Dokumenten- und Wissensverwaltung
- IR und Datenbanken
- IR und das Internet
- IR und Multimedia

Kapitel 2

Evaluierung

- Welches von mehreren IR-Systemen ist besser?
- Wie kann man die Wirkung einer Modifikation an einem IR-System bewerten?

Wozu Evaluierung?

Um verschiedene IR Systeme miteinander vergleichen zu können, müssen Evaluierungsmaße gefunden werden.

Evaluierungskriterien bei Datenbanken:

- Funktionsumfang
- im „standardisierten Bereich“: Performance (ermittelt i. Allg. durch Benchmarking)

Evaluierungskriterien bei IR-Systemen:

- Funktionsumfang / Benutzungsfreundlichkeit
- **Qualität des Ergebnisses!**
- Laufzeit- und Speicherplatzeffizienz

2.1 Effizienz ↔ Effektivität

Effizienz

- möglichst sparsamer Umgang mit den Ressourcen
- in der Informatik typisch: Laufzeit- und Speicherplatzeffizienz
- „Werden die Dinge richtig (effizient) getan?“
- Vergleichende Betrachtung nur sinnvoll bei gleichwertigen Ergebnissen

Effektivität

- bewertet die Qualität des Ergebnisses im Verhältnis zum erforderlichen Aufwand
- letztlich: „Werden die richtigen Dinge getan?“
- im IR wegen der unterschiedlichen Qualität der Ergebnisse eine sinnvolle Betrachtungsweise

Wir werden uns im folgenden zunächst auf die Bewertung der Qualität der Ergebnisse konzentrieren.

Relevanz

Bei der Evaluierung von IR-Systemen geht man davon aus, daß bezogen auf eine Anfrage q die Dokumentenkollektion in zwei Teile zerfällt:

- R_q die Menge der zur Anfrage relevanten Dokumente
- $\overline{R}_q$ die Menge der für die Anfrage nicht relevanten Dokumente

Diese Annahme führt dazu, daß ein IR-System genau R_q liefern sollte.

Man kann damit die Qualität des Ergebnisses durch die Abweichung von R_q beschreiben.

Offensichtlich verbergen sich aber hinter der Aufteilung in R_q und $\overline{R}_q$ Probleme:

- Die Aufteilung hängt vom Vorwissen des Anfragers ab!
  - Die Relevanz bestimmter Dokumente kann ggf. nur mit Vorkenntnissen erkannt werden!
  - Andererseits sind ggf. einführende Überblicksartikel nur für den Laien interessant!
- Die Relevanz von Dokumenten kann voneinander abhängen.

  So kann z.B. die Relevanz eines Dokumentes b erst durch das Lesen eines anderen Dokumentes a deutlich werden.

  Folge: Ist a im Ergebnis, so ist b relevant, sonst nicht!
- Eine zweiwertige Relevanzskala erscheint zu vereinfachend.

  Sicherlich wäre eine mehrstufige oder gar eine stetige Skala realistischer.

  Das Problem wäre aber: es wird noch schwerer eine allgemein anerkannte Einordnung der Dokumente bezüglich einer Anfrage zu finden.

Sechs Kriterien zur Evaluation von Information-Retrieval-Systemen nach [CMK66]:

- *recall*

  die Fähigkeit des Systems alle relevanten Objekte zu finden
- *precision*

  die Fähigkeit des Systems nur die Objekte zu präsentieren, die relevant sind
- *time lag*

  die ϕ Zeit zwischen dem Absetzen der Anfrage und der Bereitstellung der Antwort
- *effort*

  der intellektuelle und physische Aufwand, der vom Benutzer zur Erlangung der Antwort erforderlich ist (z.B. komplexe Eingaben, Rückfragen des Systems, ...)
- *form of presentation*

  die Qualität der Ergebnisdarstellung, die die Nützlichkeit für den Benutzer beeinflußt
- *coverage of the collection*

  der Grad, zu dem das System relevante Objekte enthält (mehr eine Frage der Dokumente als des Retrieval-Systems)

Vickery beschreibt 6 Kriterien, die er in zwei Gruppen unterteilt [Vic70]:

Kriterien, die sich auf die Verfügbarkeit von Informationen beziehen:

- *coverage* — der Anteil der möglicherweise nützlichen Literatur, der in Betracht gezogen wurde (z.B. bei mehreren verfügbaren Kollektionen: wurden alle mit potentiell relevanten Dokumenten betrachtet)
- *recall* — der Anteil der möglicherweise nützlichen Dokumente, die im Ergebnis der Anfrage enthalten sind
- *response time* — die durchschnittliche Antwortzeit

Kriterien, die sich auf die Selektivität der Antwort beziehen:

- *precision* — die Fähigkeit des Systems unnütze Dokumente zu unterdrücken
- *usability* — der Wert der ermittelten Dokumente für den Benutzer (Zuverlässigkeit, Verständlichkeit, ...)
- *presentation* — die Form, in der die Ergebnisse dem Benutzer präsentiert werden

Leistungskriterien und ihre Einflußgrößen nach [SM83]

Leistungskriterien	Einflußgrößen
Recall und Precision	lndexierungsbreite (je breiter die Indexierung, desto besser der Recall)
	Spezifizität der Indexierungssprache (je spezifischer die Indexierungssprache, desto besser die Precision)
	Recallverbessernde Eigenschaften der Indexierungssprache (Synonymerkennung, Berücksichtigung von Begriffsbeziehungen, etc.)
	Precisionsverbessernde Eigenschaften der Indexierungssprache (Gewichtung von Deskriptoren, Mehrwortbegriffe)
	Fähigkeit der Nutzer adäquate Suchanfragen zu formulieren
	Fähigkeit der Nutzer adäquate Suchstrategien zu entwickeln
Antwortzeit	Speichertyp und Speicherorganisation Suchanfragentyp
	Sitz des Informationszentrums
	Zahl der zu bearbeitenden Suchanfragen
	Zahl der gespeicherten Dokumente

Leistungskriterien und ihre Einflußgrößen nach [SM83] (fortgesetzt)

Leistungskriterien	Einflußgrößen
Nutzeraufwand	Eigenschaften der Zugriffstechnologie
	Standort der Zugriffs- und Speichermedien
	Nutzerunterstützung von Seiten des Systems und der Systembetreuer
	Umfang der nachgewiesenen Dokumente
	Art der Nutzer-System-Interaktion
	Benutzeroberfläche zur Formulierung von Suchanfragen
Ergebnispräsentation	Zugriffs- und Ausgabetechnologie
	Umfang der gespeicherten Informationen
	Dokumenttyp (Titel, Abstract oder Volltext)
Abdeckung	Art der Informationserfassung und Speichertechnologie
	Art der inhaltlichen Erschließung (bei einfacher Erschließung kann ein größerer Bereich abgedeckt werden)
	Informationsnachfrage (die lnformationsnachfrage steigt mit der Größe des abgedeckten Sachgebiets).

2.2 Recall und Precision

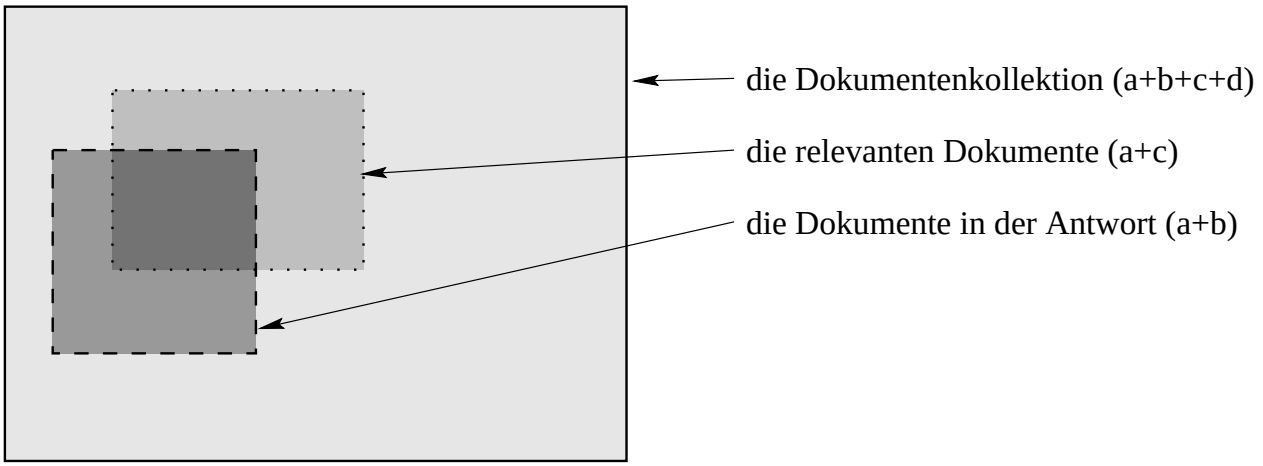
	relevant	nicht relevant	Σ
im Ergebnis	a (hits)	b (noise)	$a+b$
nicht im Ergebnis	c (misses)	d (rejected)	$c+d$
Σ	$a+c$	$b+d$	$a+b+c+d$

$$\text{Recall} = \frac{a}{a+c} = \frac{\text{Anzahl der relevanten Dokumente im Ergebnis}}{\text{Gesamtzahl der relevanten Dokumente}}$$

Der Recall beantworte die Frage: *Wie vollständig ist die Antwort?*

$$\text{Precision} = \frac{a}{a+b} = \frac{\text{Anzahl der relevanten Dokumente im Ergebnis}}{\text{Gesamtzahl der Dokumente im Ergebnis}}$$

Die Precision beantworte die Frage: *Wie genau ist die Antwort?*



a (hits) relevant und auch im Ergebnis [korrekt]

b (noise) nicht relevant und trotzdem im Ergebnis [falsch]

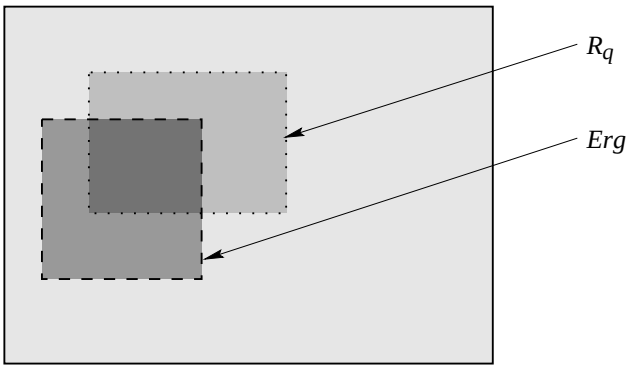
c (misses) relevant und trotzdem nicht im Ergebnis [falsch]

d (rejected) nicht relevant und auch nicht im Ergebnis [richtig]

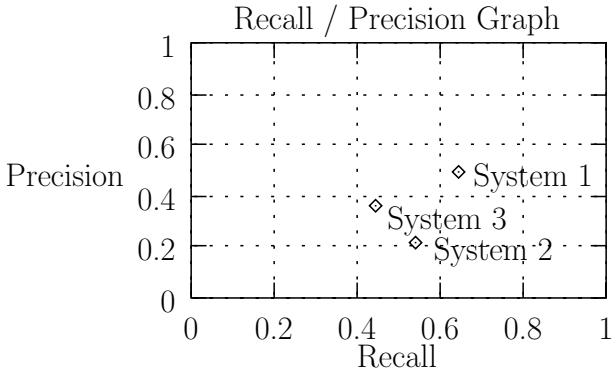
Äquivalent zu den oben gegebenen Definitionen kann man Recall un Precision auf der Basis der Mengen R_q und Erg definieren:

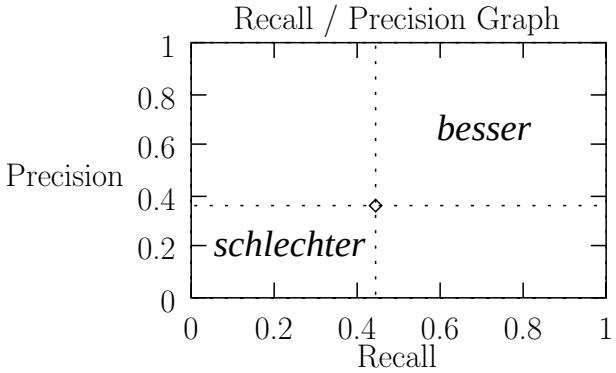
$$\text{Recall} = \frac{R_q \cap Erg}{R_q}$$

$$\text{Precision} = \frac{R_q \cap Erg}{Erg}$$



Vergleich mehrerer Systeme aufgrund von Recall/Precision-Werten





2.2.1 Bestimmung des Recall

Precision kann durch Betrachtung des Ergebnisses bestimmt werden.

Ermittlung des Recalls: Kenntnis der Gesamtheit aller relevanten Dokumente erforderlich

Folgende (Näherungs-)Methoden wurden hierzu vorgeschlagen (nach [Fuh96]):

- *Vollständige Relevanzbeurteilung einer repräsentativen Stichprobe*
  Problem:Anteil der relevanten Dokumente i. Allg. sehr gering
  $\Rightarrow$ die Stichprobe muss sehr groß sein, was zu einem großen Aufwand führt.
- *Dokument-Source-Methode*
  - man wählt ein zufälliges Dokument aus der Datenbasis
  - man formuliert eine Frage, zu der dieses Dokument relevant ist
  - Man prüft, ob das Dokument tatsächlich in der Antwort ist.
  - Über mehrere Dokumente und Anfragen gemittelt ergibt sich ein Wert für den Recall

Problem: es handelt sich nicht um „echte“ Benutzerfragen

- *Frageerweiterung*

  man erweitert die Anfrage, so dass eine Obermenge der ursprünglichen Antwortmenge gefunden wird, die wesentlich grösser ist und weitere relevante Dokumente enthält

  so kann man auch mehrere Frageformulierungen von verschiedenen Bearbeitern erstellen lassen und die Vereinigungsmenge der Antwortmengen betrachten

  Problem: Man erhält trotzdem i. Allg. nur eine Teilmenge der reelevanten Dokumente

  Folge: die Recall-Schätzungen werden im allgemeinen zu hoch sein!

- *Abgleich mit externen Quellen*

  Man versucht parallel zur Datenbanksuche noch mit davon unabhängigen Methoden, relevante Dokumente zu bestimmen

  so kann man den Fragenden oder andere Fachleute befragen, welche relevanten Dokumente sie kennen

  Der Anteil der in der Datenbasis vorhandenen Dokumente, die das System als Antwort liefert, ist dann eine gute Näherung für den Recall.

  Probleme: recht aufwendig und oft nicht anwendbar ist, weil es keine unabhängigen

externen Quellen gibt.

2.2.2 Mittelwertbildung über mehrere Anfragen

Will man über m Experimente den Mittelwert bilden, hat man zwei Möglichkeiten:

1. *Makrobewertung; nutzungsorientierter (user-oriented) Ansatz*

$$\text{Recall}_{\phi_u} = \frac{1}{m} \cdot \sum_{i=1}^{m} \frac{a_i}{a_i + c_i} \qquad \text{Precision}_{\phi_u} = \frac{1}{m} \cdot \sum_{i=1}^{m} \frac{a_i}{a_i + b_i}$$

   Hier gehen alle m Experimente unabhängig von der Ergebnisgröße gleich ein.
   Probleme gibt es, wenn einzelne Anfragen leere Ergebnisse liefern.

2. *Mikrobewertung (system-oriented)*

$$\text{Recall}_{\phi_2} = \frac{\Sigma_{i=1}^{m} a_i}{\Sigma_{i=1}^{m} (a_i + c_i)} \qquad \text{Precision}_{\phi_2} = \frac{\Sigma_{i=1}^{m} a_i}{\Sigma_{i=1}^{m} (a_i + b_i)}$$

   Hier gehen Anfragen mit einer größeren Ergebnismenge stärker ein.
   Die Summe aller Anfragen wird gewissermaßen als eine große Anfrage betrachtet.

2.2.3 Recall/Precision-Werte bei Systemen mit Ranking

bisher: System liefert nur eine unstrukturierte Antwort

jetzt: System liefert ein Ranking der Dokumente

Beispiel: $R_q = \{d_3, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{89}, d_{123}\}$

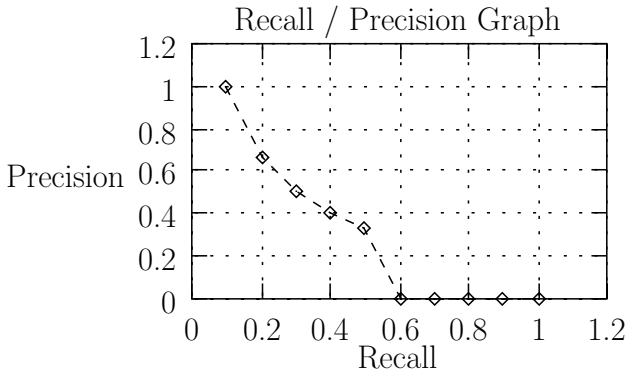
	1. d_{123} •	2. d_{84}	3. d_{56} •	4. d_6	5. d_8
Das gelieferte Ranking:	6. d_9 •	7. d_{511}	8. d_{129}	9. d_{187}	10. d_{25} •
	11. d_{38}	12. d_{48}	13. d_{250}	14. d_{113}	15. d_3 •

Da es insgesamt 10 relevante Objekte gibt und 5 davon gefunden wurden, kann man Precision-Werte für 5 Recall-Punkte angeben:

$$P\left(\frac{1}{10}\right) = 1; \quad P\left(\frac{2}{10}\right) = \frac{2}{3}; \quad P\left(\frac{3}{10}\right) = \frac{3}{6}; \quad P\left(\frac{4}{10}\right) = \frac{4}{10}; \quad P\left(\frac{5}{10}\right) = \frac{5}{15}$$

Außerdem ergibt sich für $\frac{i}{10}$ ($i \in \{6, 7, 8, 9, 10\}$) wegen des zuvor abgebrochenen Ergebnisses $P\left(\frac{i}{10}\right) = \frac{i}{\infty} = 0$.

Für das Beispiel ergibt sich:



Man beachte, daß die Verbindungslinie keine Bedeutung hat und nur der Veranschaulichung dient!

Die lineare Interpolation ergibt keine sinnvollen Precision-Werte zwischen den Recall-Punkten.

2.2.4 Probleme bei schwachen Ordnungen

Statt einer totalen Ordnung auf den Dokumenten liefern manche Systeme „nur“ eine schwache Ordnung:

- die Antwort des Systems besteht aus mehreren Rängen
- in jedem Rang können sich mehrere Dokumente befinden

Beispiel:

Die Kollektion besteht aus insgesamt 100 Dokumenten ($d_i(1 \geq i \geq 100)$).

$R_q = \{d_2, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{77}, d_{81}\}$

Wir wollen nun zwei Systeme vergleichen, die folgende Antworten liefern (relevante Dokumente sind mit einem hochgestellten r gekennzeichnet):

Rang	System 1	System2
1	$\{d_3\}$	$\{d_{41}\}$
2	$\{d_2^r, d_9^r, d_4, d_{19}, d_{20}, d_{41}, d_{55}, d_{57}, d_{92}\}$	$\{d_2^r, d_5^r, d_9^r, d_{44}^r, d_{56}^r, d_{81}^r, d_1, d_4, d_6, d_7, d_8, d_{42}, d_{55}, d_{57}, d_{92}\}$
3	$\{d_5^r, d_{44}^r, d_{71}^r, d_{77}^r\}$	$\{d_{25}^r, d_{77}^r\}$
4	$\{d_{25}^r, d_{39}^r, d_{56}^r, d_{81}^r, d_{37}, d_{48}\}$	$\{d_{39}^r, d_{71}^r\}$
5	die 80 verbleibenden Dokumente	die 80 verbleibenden Dokumente

Da es für die Beurteilung der Systeme nicht auf die konkreten Dokumente ankommt schreibt man dies auch in Form einer **Distribution**:

- System 1: `(-|++-------|++++|++++--|-[80-mal])`
- System 2: `(-|++++++---------|++|++|-[80-mal])`

`+`-Zeichen stehen dabei für relevante Dokumente.

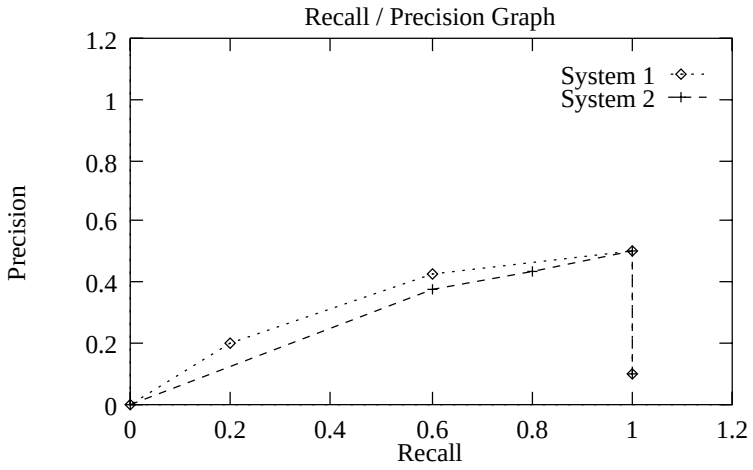
`-`-Zeichen stehen für nicht relevante Dokumente.

`|`-Zeichen trennen die Ränge.

Nach Konvention werden in jedem Rang die `+` vor die `-` geschrieben. Die Präsentationsreihenfolge im Ergebnis ist aber zufällig!

Sinnvolle Precision-Werte können zunächst nur nach den Rängen angegeben werden:

	System 1		System 2	
Rang	Recall	Precision	Recall	Precision
1	$\frac{0}{10} = 0.00$	$\frac{0}{1} = 0.00$	$\frac{0}{10} =$	$\frac{0}{1} = 0.00$
2	$\frac{2}{10} = 0.20$	$\frac{2}{10} = 0.20$	$\frac{6}{10} =$	$\frac{6}{16} = 0.38$
3	$\frac{6}{10} = 0.60$	$\frac{6}{14} = 0.43$	$\frac{8}{10} =$	$\frac{8}{18} = 0.44$
4	$\frac{10}{10} = 1.00$	$\frac{10}{20} = 0.50$	$\frac{10}{10} =$	$\frac{10}{20} = 0.50$
5	$\frac{10}{10} = 1.00$	$\frac{10}{100} = 0.10$	$\frac{10}{10} =$	$\frac{10}{100} = 0.10$



Expected Precision

Man kann die durch die Ränge entstehenden Lücken schließen und den zu erwartenden Recall nach einer vorgegebenen Anzahl von „gezogenen“ Dokumenten bestimmen.

ND	:	Anzahl der Dokumente, die „gezogenen“ werden sollen.
t_r	:	Anzahl der relevanten Dokumente in den Rängen, die vollständig gezogen werden müssen.
k	:	Anzahl der Dokumente, die aus dem Rang, in dem ND gezogene Dokumente erreicht werden, gezogen werden müssen.
r	:	Anzahl der relevanten Dokumente in dem Rang, in dem ND gezogene Dokumente erreicht werden.
i	:	Anzahl der irrelevanten Dokumente in dem Rang, in dem ND gezogene Dokumente erreicht werden.

Formal: Sei r_i die Anzahl der Dok. in Rang i und m die Anzahl der Ränge ($1 \leq i \leq m$).

t_r	:	Anzahl der relevanten Dok. in den Rängen $1, \ldots, j$ mit $\Sigma_{i=1}^{j} r_i \leq ND$.
k	:	Anzahl der Dokumente, die aus dem Rang $j+1$ gezogen werden müssen. $k = ND - \Sigma_{i=1}^{j} r_i$.
r	:	Anzahl der relevanten Dokumente in Rang $j+1$.
i	:	Anzahl der irrelevanten Dokumente in dem Rang $j+1$.

Wir erhalten:

$$EP = \frac{1}{ND}\left(t_r + k \cdot \frac{r}{r+i}\right)$$

Für System 1 erhalten wir z.B. für $ND = 17$:

$$EP = \frac{1}{17}\left(6 + 3 \cdot \frac{4}{4+2}\right)$$

Expected Recall

Um einen Recall/Precision-Graph zeichnen zu können, muß man auch einen **Expected Recall** nach ND gezogenen Dokumenten bestimmen:

Sei $|R_q|$ die Gesamtzahl der relevanten Dokumente, so erhalten wir:

$$ER = \frac{1}{|R_q|}\left(t_r + k \cdot \frac{r}{r+i}\right)$$

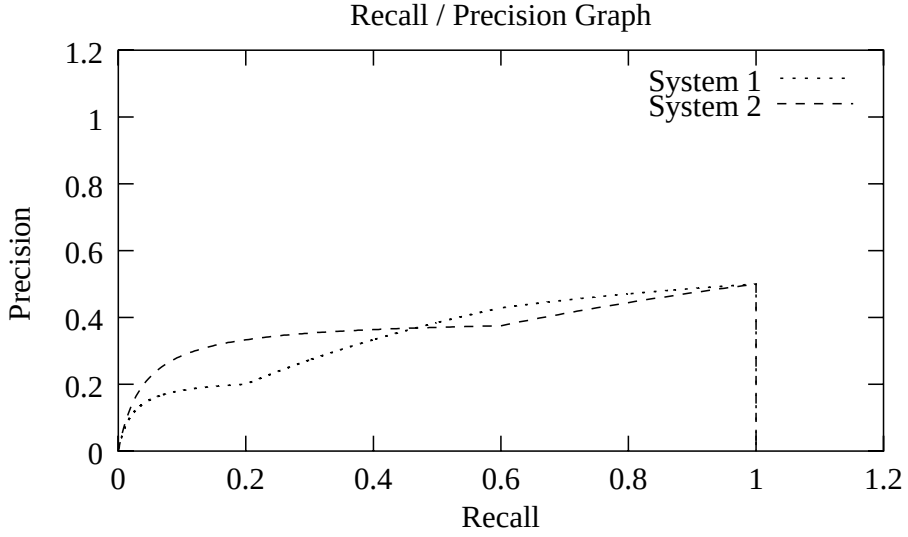
Für System 1 erhalten wir z.B. für $ND = 17$:

$$ER = \frac{1}{10}\left(6 + 3 \cdot \frac{4}{4+2}\right)$$

Für ND können dabei ohne Probleme auch nichtganzzahlige Werte verwendet werden. Für System 1 erhalten wir z.B. für $ND = 14.3$:

$$EP = \frac{1}{14.3}\left(6 + 0.3 \cdot \frac{4}{4+2}\right) \quad \text{und} \quad ER = \frac{1}{10}\left(6 + 0.3 \cdot \frac{4}{4+2}\right)$$

Recall/Precision-Graph für Expected Recall / Expected Precision



Man beachte, dass jetzt eine stetige Kurve vorliegt!

⇒ Hier hat die ganze Kurve eine sinnvolle Interpretation!

Verschiedene Abbruchkriterien

Bisher implizit: Der Nutzer bricht, die Betrachtung von Dokumenten ab, nachdem er *ND* Dokumente gesehen hat.

Andere mögliche Abbruchkriterien:

- der Benutzer bricht ab, nachdem er n relevante Dokumente gesehen hat
- der Benutzer bricht ab, nachdem er n nicht relevante Dokumente gesehen hat
- der Benutzer bricht ab, nachdem er n nicht relevante Dokumente in Folge gesehen hat

In diesen Fällen muß aber die Definition für *EP* geändert werden!

- Bisher war das letzte gezogene Dokument mit der Wahrscheinlichkeit $\frac{r}{r+i}$ ein relevantes.
- Beim ersten oben genannten alternativen Benutzerstandpunkt ist das letzte betrachtete Dokument immer ein relevantes!
- Bei den beiden zuletzt genannten alternativen Benutzerstandpunkten ist das letzte betrachtete Dokument immer ein nicht relevantes!

Berechnung der Expected Precison, wenn nach dem NR-ten relevanten Dokument abgebrochen wird

Das *NR*-te relevante Dokument kann an verschiedenen Positionen gefunden werden.

(Aber immer nur in genau einem Rang!)

⇓

wir müssen die Precision-Werte, die sich für die Positionen ergeben, mit den Wahrscheinlichkeiten dafür, dass das *NR*-te relevante Dokument an der Position gefunden wird, gewichten:

$$EP_{NR} = \sum_{\text{mögliche Position für das } NR\text{-te relevante Dok.}} (\text{Precision für diese Pos.}) \cdot (\text{Wahrscheinlichkeit der Pos.})$$

Betrachten wir die Distribution: `(-|++-------|++++|++++--|-[80-mal])`

- Der Wert $NR = 8$ wird hier im 4-ten Rang erreicht.
- Er kann innerhalb dieses Ranges an den Positionen 2, 3 oder 4 erreicht werden, weil zuvor ein anderes relevantes und danach noch zwei relevante Dokumente gezogen werden müssen.

Um die *Wahrscheinlichkeit der Pos.* auszurechnen benötigen wir einige Definitionen:

r_i	:	Anzahl der Dokumente in Rang i $(1 \leq i)$
r_i^+	:	Anzahl der relevanten Dokumente in Rang i $(1 \leq i)$
r_i^-	:	Anzahl der nicht relevanten Dokumente in Rang i $(1 \leq i)$
NR	:	Anzahl relevanten Dokumente, die der Benutzer betrachten möchte
pos	:	die Position im „aktuellen“ Rang, an der das NR-te relevante Dokument gefunden wird

Beispiel: Betrachten wir die Distribution: `(-|++-------|++++|++++--|-[80-mal])`

Wie hoch ist die Wahrscheinlichkeit, dass das 8-te relevante Dokument insgesamt als 3. Dokument aus Rang 4 gezogen wird?

P(das 3. gezogene Elem. ist relevant) * P(von den 2 davor gez. Elem. ist genau eins relevant)

4/6, da im Rang von 6 Dok. 4 relevant sind

berechnet sich nach dem Ziehen aus einer Urne ohne Zurücklegen

Aus einer mathematischen Formelsammlung [BSM97]:

> In einer Urne seien insgesamt N Kugeln, davon M weiße, $N - M$ schwarze. Es werden nacheinander ohne Zurücklegen n Kugeln gezogen. Dann ist die Wahrscheinlichkeit, daß unter diesen n Kugeln k weiße sind gleich

$$P_k(N, M, n) = \frac{\binom{M}{k}\binom{N-M}{n-k}}{\binom{N}{n}}$$

Daraus folgt in unserer Situation:

> In einem Rang x seien neben dem in seiner Position bereits festgelegten NR-ten relevanten Dokument noch $r_x - 1$ Dokumente, davon noch $r_x^+ - 1$ relevante, r_x^- nicht relevante. Es werden nacheinander ohne Zurücklegen $pos - 1$ Dokumente gezogen. Dann ist die Wahrscheinlichkeit, daß unter diesen $pos - 1$ Dokumenten k relevante sind, gleich

$$P_k(r_x, r_x^+, r_x^-, pos) = \frac{\binom{r_x^+ - 1}{k}\binom{r_x^-}{(pos-1)-k}}{\binom{r_x - 1}{pos - 1}}$$

Für unser Beispiel ergibt sich damit:

$$\text{P(von den 2 davor gez. Elem. ist genau eins relevant)} = \frac{\binom{4-1}{1}\binom{2}{(3-1)-1}}{\binom{6-1}{3-1}} = 0.6$$

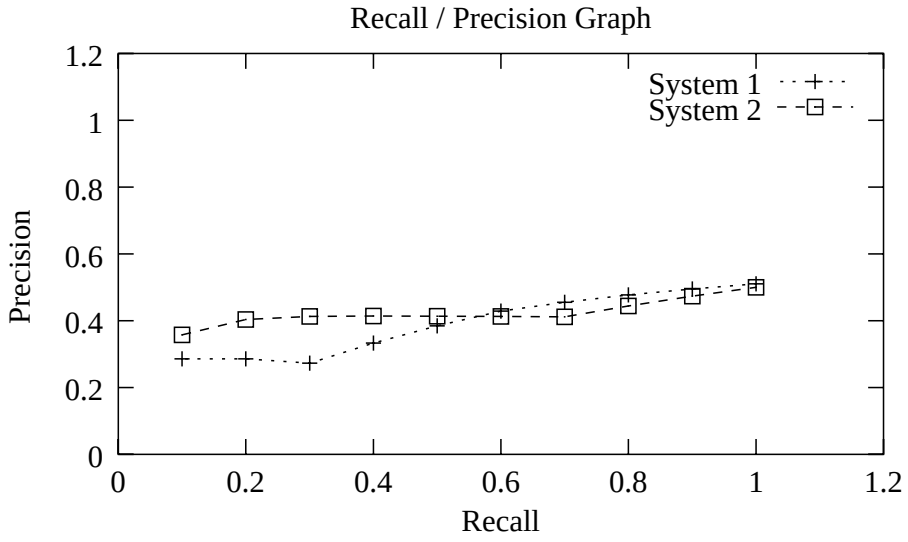
Analog erhalten wir für die Position 2 eine Wahrscheinlichkeit von 0.6 und für die Position 4 eine Wahrscheinlichkeit von 0.3.

Insgesamt ergibt sich für $NR = 8$:

$$\begin{aligned} EP_{NR} &= \Sigma_{pos\in\{2,3,4\}} \text{(Precision für diese Pos.)} \cdot \text{(Wahrscheinlichkeit der Pos.)} \\ &= \tfrac{8}{16} \cdot \tfrac{4}{6} \cdot 0.6 + \tfrac{8}{17} \cdot \tfrac{4}{6} \cdot 0.6 + \tfrac{8}{18} \cdot \tfrac{4}{6} \cdot 0.3 \\ &= 0.2 + 0.188 + 0.089 = 0.477 \end{aligned}$$

Insgesamt ergibt sich für unsere Beispieldistributionen:

- System 1: `(-|++-------|++++|+++++--|-[80-mal])`
- System 2: `(-|++++++---------|++|++|-[80-mal])`



Man beachte, dass hier die Verbindungslinien durch lineare Interpolation gebildet sind und eigentlich nicht sinnvoll interpretiert werden können.

2.2.5 Vergleich mehrerer Systeme, die ein Ranking liefern

Um mehrere Systeme miteinander zu vergleichen bestimmt man üblicherweise die Precision-Werte an vorgegebenen Recall-Punkten.

Beispiele für vorgegebene Recall-Punkte:

$$0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, (1.0) \qquad \text{oder} \qquad 0.25, 0.5, 0.75$$

Hierzu eignet sich offensichtlich am besten die Berechnung der Expected Precision unter der Annahme, dass der Benutzer nach einer vorgegebenen Anzahl von betrachteten Dokumenten abbrechen will.

Man kann die Vergleichswerte dann mittels EP und ER bestimmen.

Es ist dabei sinnvoll für jeden Recall Punkt den Mittelwert über eine Anzahl von Anfragen und Dokumentenkollektionen zu bilden.

Ob man aus den ermittelten Zahlen dann eine haltbare Aussage über die Vorteilhaftigkeit der Verfahren ableiten kann, muss mit Testverfahren aus der Statistik nachgeprüft werden.

2.3 Ein anderes Verfahren zur Bewertung: Das Nützlichkeitsmass [FMS91]

Es handelt sich um ein relatives Maß, dass eine Retrieval-Methode A und eine Retrieval-Methode B im Hinblick auf ihre Effektivität vergleicht.

Eine Retrieval-Methode x ist dabei durch die Retrieval-Funktion RSV_x gegeben.

RSV_x berechnet zu einer Anfrage q und einem Dokument d den *Retrieval Status Value* $\text{RSV}_x(q, d)$.

Die Dokumente werden dann im Ergebnis absteigend nach den $\text{RSV}_x(q, d)$ geordnet.

Ferner geht man davon aus, dass der Benutzer nicht mehr als $2r$ Dokumente „sehen“ will.

Für eine gegebene Dokumentenkollektion D, eine Anfrage q, und einen Schwellenwert r definieren wir die Antwortmenge $R_x(D, q, r)$ der Retrieval-Methode x wie folgt:

> Sofern D mehr als r Dokumente enthält, besteht $R_x(D, q, r)$ aus den r Dokumenten mit den höchsten Werten für RSV_x. Anderenfalls aus allen.

Da wir zwei Retrieval-Methoden miteinander vergleichen, muss der Benutzer die Vereinigung der beiden Ergebnismengen betrachten:

$$R(D, q, r) = R_A(D, q, r) \cup R_B(D, q, r)$$

$R(D, q, r)$ enthält dabei offensichtlich höchstens $2r$ Dokumente.

Sofern $r \leq |D|$ enthält $R(D, q, r)$ mindestens r Dokumente, und
sofern $r \geq |D|$ ist gilt $R(D, q, r) = D$.

Zum Vergleich der Methoden muss der Benutzer nun eine Entscheidung über die Relevanz der Dokumente machen.

Hierzu muss aber – im Gegensatz zu klassischen Verfahren – nicht die gesamte Dokumentenkollektion in relavante und nicht relevante Dokumente unterteilt werden.

Statt dessen müssen nur relative Relevanzurteile abgegeben werden
(Dokument d ist relevanter als Dokument d').

Ferner werden solche Urteile nur für Dokumente aus $R(D, q, r)$ benötigt!

Die Relevanzbeurteilungen werden dabei durch eine *Präferenzrelation* $<_p$ beschrieben, wobei p für den aktuellen Betrachter steht.

$d <_p d'$ bedeutet: Benutzer p hält Dokument d im Hinblick auf die gegebene Anfrage für weniger nützlich als Dokument d'.

Wir definieren die Präferenzen des Benutzers als

$$\pi_p = \{(d, d') \,|\, d <_p d'\}$$

Dabei interessieren uns wie gesagt nur die Dokumente in $R(D, q, r)$, also $\pi_p \cap R^2(D, q, r)$.

$(R^2(D, q, r) = R(D, q, r) \times R(D, q, r))$

Man beachte, dass π_p im allgemeinen eine partielle Ordnung ist:

$\Rightarrow$ nicht für alle Paare aus $R^2(D, q, r)$ gibt es eine klare Relation.

Des weiteren definieren wir für die Retrieval-Methode x ($x \in \{A, B\}$) die sich aus den RSV-Werten ergebenden Präferenzen:

$$\pi_x = \{(d, d') \,|\mathrm{RSV}_x(q, d) < \mathrm{RSV}_x(q, d')\}$$

Wir führen nun zwei Zufallsvariablen $X(D, p, q, r)$ und $Y(D, p, q, r)$ ein:

$$X(D, p, q, r) = \frac{\left|R^2(D, q, r) \cap \pi_p \cap \pi_A\right| - \left|R^2(D, q, r) \cap \pi_p^{-1} \cap \pi_A\right|}{\left|R^2(D, q, r) \cap \pi_p\right|}$$

$$Y(D, p, q, r) = \frac{\left|R^2(D, q, r) \cap \pi_p \cap \pi_B\right| - \left|R^2(D, q, r) \cap \pi_p^{-1} \cap \pi_B\right|}{\left|R^2(D, q, r) \cap \pi_p\right|}$$

Wir nehmen also jeweils die Übereinstimmungen zwischen der Retrieval-Methode und den Benutzerpräferenzen und ziehen von diesen die Widersprüche ab.

Das Ergebnis teilen wir durch die Anzahl der Benutzerpräferenzen.

π_p^{-1} sei dabei definiert durch $\pi_p^{-1} = \{(d, d') \,|(d', d) \in \pi_p\}$.

Im Gegensatz zu den Zufallsvariablen $X(D, p, q, r)$ und $Y(D, p, q, r)$ bezeichnen wir die aktuellen Werte aus den Experimenten mit $x(D, p, q, r)$ und $y(D, p, q, r)$.

Der Wert $u_{A,B}$, der die Nützlichkeit von B relativ zu A angibt, wird nun in k Experimenten ermittelt.

Jedes Experiment entspricht dabei einem Ereignis (D_i, p_i, q_i, r_i) mit $0 \leq i < k$.

$u_{A,B}$ gibt – grob gesprochen – an, wie oft die Werte $x(D_i, p_i, q_i, r_i)$ im Durchschnitt kleiner sind als die Werte $y(D_i, p_i, q_i, r_i)$.

Im weiteren kürzen wir $x(D_i, p_i, q_i, r_i)$ mit x_i ab, und $y(D_i, p_i, q_i, r_i)$ mit y_i.

Für die k Ereignisse (D_i, p_i, q_i, r_i) werden nun die Werte x_i und y_i nach den obigen Formeln für $X(D, p, q, r)$ und $Y(D, p, q, r)$ berechnet.

Ein Beispiel für das Nützlichkeitsmaß bis hierher

Wir vergleichen zwei Methoden A und B, für die je vier Anfragen durchgeführt werden.

Die folgende Tabelle zeigt die **Ergebnisse der Anfragen**, wir gehen von $r = 4$ aus:

	A				B			
Rang	q_0	q_1	q_2	q_3	q_0	q_1	q_2	q_3
1	d_3	d_4	d_7	d_6	d_2	d_4	d_6	d_9
2	d_2	d_5	d_6	d_7	d_3	d_5	d_7	d_7
3	d_0	d_3	d_8	d_8	d_1	d_3	d_8	d_6
4	d_1	d_7		d_9	d_0	d_6		d_8
5	d_4	d_6			d_4	d_7		

Wegen $r = 4$ werden nur die ersten vier Zeilen der Tabelle betrachtet.

Die Antwortmengen $R(D, q, r) = R_A(D, q, r) \cup R_B(D, q, r)$ ergeben sich damit zu:

Anfrage	$R(D, q, r)$
q_0	$\{d_0, d_1, d_2, d_3\}$
q_1	$\{d_3, d_4, d_5, d_6, d_7\}$
q_2	$\{d_6, d_7, d_8\}$
q_3	$\{d_6, d_7, d_8, d_9\}$

Die Präferenzen des Benutzers seien wie folgt:

Anfrage	$\pi_p \cap R^2(D, q, r)$
q_0	$\{(d_0, d_1), (d_1, d_2)\}$
q_1	$\{(d_3, d_4)\}$
q_2	$\{(d_6, d_7)\}$
q_3	$\{(d_6, d_7), (d_7, d_9)\}$

Die erfüllten Präferenzen ergeben sich damit zu:

Anfrage	$\pi_A \cap \pi_p \cap R^2(D,q,r)$	$\pi_B \cap \pi_p \cap R^2(D,q,r)$
q_0	$\{(d_1,d_2)\}$	$\{(d_0,d_1),(d_1,d_2)\}$
q_1	$\{(d_3,d_4)\}$	$\{(d_3,d_4)\}$
q_2	$\{(d_6,d_7)\}$	$\{\}$
q_3	$\{\}$	$\{(d_6,d_7),(d_7,d_9)\}$

Die verletzten Präferenzen ergeben sich damit zu:

Anfrage	$\pi_A \cap \pi_p^{-1} \cap R^2(D,q,r)$	$\pi_B \cap \pi_p^{-1} \cap R^2(D,q,r)$
q_0	$\{(d_1,d_0)\}$	$\{\}$
q_1	$\{\}$	$\{\}$
q_2	$\{\}$	$\{(d_7,d_6)\}$
q_3	$\{(d_7,d_6),(d_9,d_7)\}$	$\{\}$

Die Werte für x_i und y_i berechnen sich nach den folgenden Formeln (wir gehen davon aus, dass die Dokumentenkollektion D, der Benutzer p und der Schwellenwert r für alle Anfragen gleich sind):

$$x_i = x(D,p,q_i,r) = \frac{\left|R^2(D,q_i,r) \cap \pi_p \cap \pi_A\right| - \left|R^2(D,q_i,r) \cap \pi_p^{-1} \cap \pi_A\right|}{\left|R^2(D,q_i,r) \cap \pi_p\right|}$$

$$y_i = y(D,p,q_i,r) = \frac{\left|R^2(D,q_i,r) \cap \pi_p \cap \pi_B\right| - \left|R^2(D,q_i,r) \cap \pi_p^{-1} \cap \pi_B\right|}{\left|R^2(D,q_i,r) \cap \pi_p\right|}$$

Damit ergibt sich:

Anfrage	x_i	y_i
q_0	$\frac{1-1}{2} = 0$	$\frac{2-0}{2} = 1$
q_1	$\frac{1-0}{1} = 1$	$\frac{1-0}{1} = 1$
q_2	$\frac{1-0}{1} = 1$	$\frac{0-1}{1} = -1$
q_3	$\frac{0-2}{2} = -1$	$\frac{2-0}{2} = 2$

Wir betrachten nun weiter das Verfahren und dazu jeweils das Beispiel

Die **Summe der positiven Ränge** w_+ ergibt sich dann wie folgt:

1. Berechne die Differenzen $y_i - x_i$ und streiche die Differenzen, die gleich 0 sind. (Sie können nicht zum Vergleich von A und B beitragen.)

Anfrage	x_i	y_i	$y_i - x_i$	streichen?
q_0	$\frac{1-1}{2} = 0$	$\frac{2-0}{2} = 1$	1	nein
q_1	$\frac{1-0}{1} = 1$	$\frac{1-0}{1} = 1$	0	ja
q_2	$\frac{1-0}{1} = 1$	$\frac{0-1}{1} = -1$	-2	nein
q_3	$\frac{0-2}{2} = -1$	$\frac{2-0}{2} = 1$	2	nein

2. Ordne die absoluten Werte $|y_i - x_i|$ absteigend an.
   Falls es „Cluster“ gibt ($|y_i - x_i| = |y_j - x_j|$) wird jedem Element des Clusters der Durchschnittliche Rang der Clusterelemente zugeordnet.

Anfrage	$\lvert y_i - x_i \rvert$	Rangzahl
q_0	1	1
q_2	2	2.5
q_3	2	2.5

3. Jetzte werden die Vorzeichen der $|y_i - x_i|$ wieder zu den Rängen hinzugefügt.
   Wir erhalten „vorzeichenbehaftete Ränge“.

Anfrage	$\lvert y_i - x_i \rvert$	Vorzeichen	Rangzahl
q_0	1	+	1
q_2	2	−	2.5
q_3	2	+	2.5

4. w_+ ergibt sich nun als die **Summe der Rangzahlen zu den Rängen mit positivem Vorzeichen**.

   Im Beispiel ergibt sich $w_+ = 1 + 2.5 = 3.5$ als Summe der Rangzahlen zu den Rängen mit positivem Vorzeichen.

Sofern die Methode B durchgängig besser als A ist, werden viele Differenzen mit positivem Vorzeichen hohe Ränge einnehmen.

Dadurch ist w_+ hoch, sofern B durchgängig besser als A.

Man beachte aber, dass ein hoher Wert für w_+ nichts darüber aussagt, ob B viel besser, oder nur ein wenig besser als A ist.

Die Nützlichkeit $u_{A,B}$ wird nun definiert als die normalisierte Abweichung von w_+ zu μ.

μ ist der Erwartungswert von W_+, der sich ergibt, wenn X_i und Y_i die gleiche Verteilung haben.

Die Anzahl k wird dabei auf die Anzahl k_0 der Experimente mit $y_i - x_i \neq 0$ reduziert.

$$u_{A,B} = \frac{w_+ - \mu}{\mu} \qquad \mu = \frac{k_0(k_0+1)}{4}$$

$u_{A,B}$ gibt dabei an, wie oft im Durchschnitt die Werte y_i größer sind als die Werte x_i.

Im Beispiel ergibt sich wegen $k_0 = 3$ folgendes: $\mu = \frac{k_0(k_0+1)}{4} = \frac{3(3+1)}{4} = 3$.

Um einen Eindruck davon zu bekommen, wieviel größer die y_i im Vergleich zu den x_i sind, können wir eine angepasste Nützlichkeit definieren, die auch die 0 Differenzen enthält:

$$u^*_{A,B} = u_{A,B} \frac{1}{k} \left| \sum_{i=0}^{k-1} (y_i - x_i) \right|$$

Hier wirkt sich nun auch die Größe der Differenzen aus.

Da $u^*_{A,B}$ statistisch schwer zu handhaben ist, betrachten wir im weiteren wieder $u_{A,B}$.

Sofern A effektiver ist als B ist der Erwartungswert für $U_{A,B}$ negativ.

Ein negativer Wert für $u_{A,B}$ deutet somit an, dass A effektiver sein könnte als B.

Andererseits könnte aber obwohl A effektiver ist als B auch zufällig durch eine „unglückliche“ Wahl der Beispielanfragen ein positiver Wert entstehen.

Dieser positive Wert könnte uns dann fälschlich glauben lassen, B sei effektiver als A.

Sei $u_{A,B}$ ein positiver Wert, der sich aus k Experimenten ergeben hat und andeutet, daß B effektiver ist als A.

Dann bedeutet eine geringe Wahrscheinlichkeit

$$P_k(U_{A,B} \geq u_{A,B})$$

dass es unwahrscheinlich ist, dass $u_{A,B}$ eine höhere Effektivität von B nahelegt, obwohl in Wirklichkeit A effektiver ist als B oder zumindest gleichwertig.

Der Wert $P_k(U_{A,B} \geq u_{A,B})$ wird dabei als „Fehlerwahrscheinlichkeit“ bezeichnet.

$P_k(U_{A,B} \geq u_{A,B})$ ist die Fehlerwahrscheinlichkeit dafür, dass man die Hypothese dass A mindestens so effektiv ist wie B verwirft, obwohl sie eigentlich wahr ist.

Ein Ablehnen der Hypothese, dass A mindestens so effektiv ist wie B, bedeutet aber, dass wir B für effektiver als A halten.

Sofern $k \geq 20$, kann die Fehlerwahrscheinlichkeit wie folgt berechnet werden:

$$P_k(U_{A,B} \geq u_{A,B}) \approx 1 - \Phi\left(\frac{w_+ - \mu}{\sigma}\right)$$

mit

$$\mu = \frac{k_0(k_0+1)}{4} \qquad \sigma^2 = \frac{k_0(k_0+1)(2k_0+1)}{24}$$

$\Phi(z)$ gibt dabei die kummulative Verteilungsfunktion der Normalverteilung an:

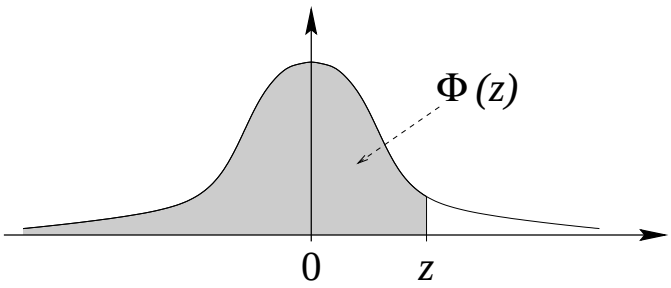


Tabelle für $\Phi(z)$:

z	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9
0,0...	0,5000	0,5040	0,5080	0,5120	0,5160	0,5199	0,5239	0,5279	0,5319	0,5359
0,1...	0,5398	0,5438	0,5478	0,5517	0,5557	0,5596	0,5636	0,5675	0,5714	0,5753
0,2...	0,5793	0,5832	0,5871	0,5910	0,5948	0,5987	0,6026	0,6064	0,6103	0,6141
0,3...	0,6179	0,6217	0,6255	0,6293	0,6331	0,6368	0,6406	0,6443	0,6480	0,6517
0,4...	0,6554	0,6591	0,6628	0,6664	0,6700	0,6736	0,6772	0,6808	0,6844	0,6879
0,5...	0,6915	0,6950	0,6985	0,7019	0,7054	0,7088	0,7123	0,7157	0,7190	0,7224
0,6...	0,7257	0,7291	0,7324	0,7357	0,7389	0,7422	0,7454	0,7486	0,7517	0,7549
0,7...	0,7580	0,7611	0,7642	0,7673	0,7704	0,7734	0,7764	0,7794	0,7823	0,7852
0,8...	0,7881	0,7910	0,7939	0,7967	0,7995	0,8023	0,8051	0,8078	0,8106	0,8133
0,9...	0,8159	0,8186	0,8212	0,8238	0,8264	0,8289	0,8315	0,8340	0,8365	0,8389
1,0...	0,8413	0,8438	0,8461	0,8485	0,8508	0,8531	0,8554	0,8577	0,8599	0,8621
1,1...	0,8643	0,8665	0,8686	0,8708	0,8729	0,8749	0,8770	0,8790	0,8810	0,8830
1,2...	0,8849	0,8869	0,8888	0,8907	0,8925	0,8944	0,8962	0,8980	0,8997	0,9015
1,3...	0,9032	0,9049	0,9066	0,9082	0,9099	0,9115	0,9131	0,9147	0,9162	0,9177
1,4...	0,9192	0,9207	0,9222	0,9236	0,9251	0,9265	0,9279	0,9292	0,9306	0,9319
1,5...	0,9332	0,9345	0,9357	0,9370	0,9382	0,9394	0,9406	0,9418	0,9429	0,9441
1,6...	0,9452	0,9463	0,9474	0,9484	0,9495	0,9505	0,9515	0,9525	0,9535	0,9545
1,7...	0,9554	0,9564	0,9573	0,9582	0,9591	0,9599	0,9608	0,9616	0,9625	0,9633
1,8...	0,9641	0,9649	0,9656	0,9664	0,9671	0,9678	0,9686	0,9693	0,9699	0,9706
1,9...	0,9713	0,9719	0,9726	0,9732	0,9738	0,9744	0,9750	0,9756	0,9761	0,9767
2,0...	0,9772	0,9778	0,9783	0,9788	0,9793	0,9798	0,9803	0,9808	0,9812	0,9817
2,1...	0,9821	0,9826	0,9830	0,9834	0,9838	0,9842	0,9846	0,9850	0,9854	0,9857
2,2...	0,9861	0,9864	0,9868	0,9871	0,9875	0,9878	0,9881	0,9884	0,9887	0,9890
2,3...	0,9893	0,9896	0,9898	0,9901	0,9904	0,9906	0,9909	0,9911	0,9913	0,9916
2,4...	0,9918	0,9920	0,9922	0,9925	0,9927	0,9929	0,9931	0,9932	0,9934	0,9936
2,5...	0,9938	0,9940	0,9941	0,9943	0,9945	0,9946	0,9948	0,9949	0,9951	0,9952
2,6...	0,9953	0,9955	0,9956	0,9957	0,9959	0,9960	0,9961	0,9962	0,9963	0,9964
2,7...	0,9965	0,9966	0,9967	0,9968	0,9969	0,9970	0,9971	0,9972	0,9973	0,9974
2,8...	0,9974	0,9975	0,9976	0,9977	0,9977	0,9978	0,9979	0,9979	0,9980	0,9981
2,9...	0,9981	0,9982	0,9982	0,9983	0,9984	0,9984	0,9985	0,9985	0,9986	0,9986
3,0...	0,9986	0,9987	0,9987	0,9988	0,9988	0,9989	0,9989	0,9989	0,9990	0,9990
3,1...	0,9990	0,9991	0,9991	0,9991	0,9992	0,9992	0,9992	0,9992	0,9993	0,9993
3,2...	0,9993	0,9993	0,9994	0,9994	0,9994	0,9994	0,9994	0,9995	0,9995	0,9995

Noch einmal zurück zu unserem Beispiel:

- Wegen $k_0 = 3$ ergab sich $\mu = \frac{k_0(k_0+1)}{4} = \frac{3(3+1)}{4} = 3$.
- Ferner erhalten wir $\sigma^2 = \frac{k_0(k_0+1)(2k_0+1)}{24} = \frac{3(3+1)(2\cdot 3+1)}{24} = 3.5$.
- $u_{A,B} = \frac{w_+ - \mu}{\mu}$ ergibt sich dann zu $\frac{3.5-3}{3} = 0.166\ldots$.
- $P_k(U_{A,B} \geq u_{A,B}) \approx 1 - \Phi\left(\frac{w_+ - \mu}{\sigma}\right) = 1 - \Phi\left(\frac{0,5}{\sqrt{3.5}}\right) = 1 - \Phi\left(0.26726\right) \approx 1 - 0,604 = 0,396$
- Damit haben wir (wegen des kleinen Beispiels naheliegenderweise) keinen fundierten Anhalt, um die These zu verwerfen, dass A besser als B ist.

Einordnung des Nützlichkeitsmaßes

Das Nützlichkeitsmaß konzentriert sich auf die subjektive Nützlichkeitsbeurteilung der einzelnen Benutzer.

Ferner werden im Gegensatz zu klassischen Verfahren nur relative Urteile benötigt.

Mit anderen Worten: der Benutzer muß nur eine persönliche Relevanzordnung angeben, die keine totale Ordnung sein muss.

Dies fällt dem Benutzer i. Allg. leicher als eine Aufteilung in relevante und nicht relevante Dokumente.

Ein weiterer Vorteil des Nützlichkeitsmasses ist seine statistische Fundierung.

Man kann über die Fehlerwahrscheinlichkeit die „Sicherheit“ der getroffenen Aussage einordnen.

2.4 Experimente und Testkollektionen

Idee: vordefinierte Sammlungen von Dokumenten und Anfragen schaffen, an denen IR-Systeme verglichen werden können.

Mit Sammlungen können Evaluierungsmaße ermittelt werden.

Hauptprobleme liegen darin, dass

- die Sammlungen i. Allg. eher klein sind,
- häufig unklar ist, wie sie zustande gekommen sind,
- viele Sammlungen inzwischen recht alt und damit nicht mehr repräsentativ für die heutigen Inhalte und Dokumentformate sind
- die Sammlungen immer wieder verwendet werden und damit eine Optimierung der Systeme auf die Sammlungen hin stattfindet
- unterschiedliche Sammlungen sich unterschiedlich verhalten
- die Auswahl einer Sammlung für eine Evaluierung teilweise recht willkürlich erscheint (Auswahl der Sammlung, mit der die besten Ergebnisse erzielt wurden?)

2.4.1 Die TREC-Kollektion

IR-Systeme werden i. Allg. aus zwei Richtungen angegriffen:

1. Die zum Teil fehlende formale Fundierung wird kritisiert.
2. Das Fehlen robuster und konsistenter Evaluierungsumgebungen und Benchmarks.

Wie wir sehen werden, kann der erste Aspekt zum Teil durch eine entsprechende Fundierung beim probabilistischen IR angegangen werden.

Ein Ansatz zur Überwindung des zweiten Kritikpunktes ist TREC.

TREC = Text REtrieval Conference

Über drei Jahrzehnte hinweg wurden im IR Experimente mit kleinen Testkollektionen gemacht.

Diese waren kaum vergleichbar, weil jede Forschergruppe ihre eigenen Kollektionen hatte.

Anfang der 90er wurde am NIST unter der Leitung von Donna Harman ein Versuch zu einer übergreifenden Testumgebung gestartet.

NIST = *National Institute of Standards and Technology* in Maryland, USA

Dazu wurde 1992 eine jährlich stattfindende Konferenz ins Leben gerufen.

Ziel waren vergleichbare Experimente mit einer umfangreichen Kollektion von über 1 Mio. Dokumente.

Dazu wird für jede TREC Konferenz eine Menge von Referenz-Experimenten entwickelt.

Diese Experimente werden als Tasks bezeichnet. Dabei kann es z.B. darum gehen:

- zu gegebenen Anfragen mit beliebigem Aufwand ein Ranking der 1000 relevantesten Dokumente zu erstellen (*ad hoc task*), oder
- innerhalb von 5 Minuten 15 Dokumente zu selektieren, die möglichst alle relevant sein sollen (*high precision task*).

Die Forschergruppen, die an der Konferenz teilnehmen wollen, müssen mit ihren Systemen die Referenz-Experimente durchführen.

Die Ziele der Konferenz werden wie folgt formuliert [VH98]:

- die Forschung an Retrieval-Systemen für große Test-Kollektionen zu unterstützen;
- die Kommunikation zwischen der Industrie, der akademischen Welt und der Verwaltung durch ein offenes Forum für den Ideenaustausch zu unterstützen;
- den Transfer von Technologie aus den Forschungslabors in kommerzielle Produkte durch die Demonstration substantieller Fortschritte bei Retrieval-Methoden für Realweltprobleme zu beschleunigen;
- die Verfügbarkeit von angemessenen Evaluierungstechniken für die Industrie und die akademische Welt zu erhöhen (einschließlich der Entwicklung neuer Evaluierungstechniken).

Die Teilnahme an der Konferenz ist auf diejenigen beschränkt, die die TREC-Retrievalaufgaben gelöst haben.

(Ausnahmen gelten lediglich für ausgewähltes Personal der Sponsoren.)

Die Teilnehmer können dabei verschiedenste Retrieval-Techniken anwenden:

- automatische Thesauri
- spezielle Termgewichtungsfunktionen
- Verarbeitung natürlicher Sprache
- Relevance Feedback
- spezielle Verfahren der Zeichenkettensuche

Alle Verfahren werden auf die gleiche 2 GB umfassende Datenkollektion angewendet.

Dabei sind vorgegebene Informationswünsche (**topics**) zu bearbeiten.

Die Ergebnisse werden dann miteinander verglichen.

Die erste TREC Konferenz war im November 1992.

Die Konferenzen finden jährlich in Gaithersburg, Maryland USA statt.

Es gibt jedesmal neue Testkollektionen mit neuen Schwerpunkten.

An TREC-7 teilnehmende Organisationen [VH98]:

ACSys Cooperative Research Centre	Management Information Technologies, Inc.
AT&T Labs Research	Massachusetts Institute of Technology
Avignon CS Laboratory/Bertin	National Tsing Hua University
BBN Technologies	NEC Corp. and Tokyo Institute of Technology
Canadian Imperial Bank of Commerce	New Mexico State University
Carnegie Mellon University	NTT DATA Corporation
Commissariat à l'Energie Atomique	Okapi Group (City U./U. of Sheffield/Microsoft)
CLARITECH Corporation	Oregon Health Sciences University
Cornell University/SabIR Research, Inc.	Queens College, CUNY
Defense Evaluation and Research Agency	RMIT/Univ. of Melbourne/CSIRO
Eurospider	Rutgers University (2 groups)
Fondazione Ugo Bordoni	Seoul National University
FS Consulting, Inc.	Swiss Federal Institute of Technology (ETH)
Fujitsu Laboratories, Ltd.	TextWise, Inc.
GE/Rutgers/SICS/Helsinki	TNO-TPD TU-Delft
Harris Information Systems Division	TwentyOne
IBM — Almaden Research Center	Universite de Montreal
IBM T.J. Watson Research Center (2 groups)	University of California, Berkeley
Illinois Institute of Technology	University of Cambridge
Imperial College of Science, Technology and Medicine	University of Iowa
Institut de Recherche en Informatique de Toulouse	University of Maryland
The Johns Hopkins University — APL	University of Massachusetts, Amherst
Kasetsart University	University of North Carolina, Chapel Hill
KDD R&D Laboratories	Univ. of Sheffield/Cambridge/SoftSound
Keio University	University of Toronto
Lexis-Nexis	University of Waterloo
Los Alamos National Laboratory	U.S. Department of Defense

Woraus besteht die TREC Kollektion?

Die Kollektion besteht aus:

- den Dokumenten im Umfang von insgesamt 5,8 GB (TREC-7),
- derzeit insgesamt 400 *topics* – jedes Jahr kommen 50 neu hinzu,
- zu jedem *topic* gibt es eine von Experten manuell ausgewählte Menge von relevanten Dokumenten.

Die Dokumente werden auf insgesamt 5 CDs verfügbar gemacht.

Die CDs 3 und 4, die die Dokumente für die Standardaufgaben von TREC-7 enthalten, können für je 200 USD beim NIST bestellt werden.

Die *topics* und die Relevanzurteile können von der WWW-Seite `trec.nist.gov` geladen werden.

Die Dokumentenkollektion

Die Dokumente werden bei TREC auf CDs mit je ca. 1 GB komprimierten Daten zur Verfügung gestellt.

Bei TREC-7 wurden 5 CD verwendet.

Die Dokumente wurden dabei in einem SGML-Format abgelegt.

Dabei wurden für die einzelnen Datenmengen jeweils die gleichen übergeordneten Strukturen verwendet.

In den untergeordneten Details unterscheiden sich die Kollektionen aber.

Die Idee dabei ist, die Daten so nah wie möglich beim Orginattext zu halten.

Daher wird auch nicht versucht, Schreibfehler oder merkwürdige Formatierungen oder ähnliches zu korrigieren.

Die `<DOCNO>` ermöglicht das einfache Referenzieren von Dokumenten, z.B. in den Relevanzbeurteilungen.

Statistik zur Dokumentenkollektion von TREC-7 [VH98]

	Size (megabytes)	# Docs	Median # Words/Doc	Mean # Words/Doc
Disk 1				
Wall Street Journal, 1987-1989	267	98732	245	434,0
Associated Press newswire, 1989	254	84678	446	473,9
Computer Selects articles, Ziff-Davis	242	75180	200	473,0
Federal Register, 1989	260	25960	391	1315,9
abstracts of U.S. DOE publications	184	226087	111	120,4
Disk 2				
Wall Street Journal, 1990-1992 (WSJ)	242	74520	301	508,4
Associated Press newswire (1988) (AP)	237	79919	438	468,7
Computer Selects articles, Ziff-Davis (ZIFF)	175	56920	182	451,9
Federal Register (1988) (FR88)	209	19860	396	1378,1
Disk 3				
San Jose Mercury News, 1991	287	90257	379	453,0
Associated Press newswire, 1990	237	78321	451	478,4
Computer Selects articles, Ziff-Davis	345	161021	122	295,4
U.S. patents, 1993	243	6711	4445	5391,0
Disk 4				
the Financial Times, 1991-1994 (FT)	564	210158	316	412,7
Federal Register, 1994 (FR94)	395	55630	588	644,7
Congressional Record, 1993 (CR)	235	27922	288	1373,5
Disk 5				
Foreign Broadcast Information Service (FBIS)	470	130471	322	543,6
the LA Times	475	131896	351	526,5

Beispiel für ein Dokument (aus der Financial Times):

```
<DOC>
<DOCNO>FT911-3</DOCNO>
<PROFILE>AN-BEOA7AAIFT</PROFILE>
<DATE>910514</DATE>
<HEADLINE>
FT 14 MAY 91 / International Company News: Contigas plans DM900m east German
project
</HEADLINE>
<BYLINE> By DAVID GOODHART </BYLINE>
<DATELINE> BONN </DATELINE>
<TEXT>
CONTIGAS, the German gas group 81 per cent owned by the utility Bayernwerk, said
yesterday that it intends to invest DM900m (Dollars 522m) in the next four years
to build a new gas distribution system in the east German state of
Thuringia. ...
</TEXT>
</DOC>
```


Dokumentmengen und topics für die TREC-Konferenzen

TREC	Task	Documents	Topics
TREC-1	ad hoc	disks 1 & 2	51 - 100
	routing	disk 2	1 - 50
TREC-2	ad hoc	disks 1 & 2	101 - 150
	routing	disk 3	51 - 100
TREC-3	ad hoc	disks 1 & 2	151 - 200
	routing	disk 3	101 - 150
TREC-4	ad hoc	disks 2 & 3	201 - 250
	routing	CS+FR	assorted
TREC-5	ad hoc	disks 2 & 4	251 - 300
	routing	FBIS-1	assorted
TREC-6	ad hoc	disks 4 & 5	301 - 350
	routing	FBIS-2	assorted
TREC-7	ad hoc	disks 4 & 5	351 - 400
		(no CR)	

Die Anfragen (topics)

Bei den neueren TREC-Konferenzen bestehen die *topics* aus

- title (wenige das Topic beschreibende Worte)
- description (die ursprüngliche Benutzeranfrage; i. Allg. ein Satz)
- narrative (Langtext, der beschreibt, was ein Dokument relevant macht)

Mit jeder TREC-Konferenz werden 50 neue Topics ausgegeben.

Durch Verwendung der unterscheidlichen Teile der Topic-Beschreibung kann dabei der Effekt unterschiedlicher Anfragelängen auf die Performance untersucht werden.

Die Veränderungen von Konferenz zu Konferenz gehen auf rege Diskussionen zurück.

So sollen die 1 bis 3 Worte langen Titel bei TREC-7 Anfragen im WWW annähern. Sie bestehen aus bis zu drei Wörtern, die das Thema möglichst gut beschreiben.

Die Umsetzung vom *topic*-Text in eine Anfrage muss vom System geleistet werden.

Umfang der *topic*-Definitionen in Wörten:

	Min	Max	Mean
TREC-1 (51-100)	44	250	107.4
title	1	11	3.8
description	5	41	17.9
narrative	23	209	64.5
concepts	4	111	21.2
TREC-2 (101-150)	54	231	130.8
title	2	9	4.9
description	6	41	18.7
narrative	27	165	78.8
concepts	3	88	28.5
TREC-3 (151-200)	49	180	103.4
title	2	20	6.5
description	9	42	22.3
narrative	26	146	74.6
TREC-4 (201-250)	8	33	16.3
description	8	33	16.3
TREC-5 (251-300)	29	213	82.7
title	2	10	3.8
description	6	40	15.7
narrative	19	168	63.2
TREC-6 (301-350)	47	156	88.4
title	1	5	2.7
description	5	62	20.4
narrative	17	142	65.3
TREC-7 (351-400)	31	114	57.6
title	1	3	2.5
description	5	34	14.3
narrative	14	92	40.8

Beispiel für ein „Topic“aus TREC-1:

```
<top>

<head> Tipster Topic Description

<num> Number: 089

<dom> Domain: International Economics

<title> Topic: "Downstream" Investments by OPEC Member States

<desc> Description:
Document must identify an existing or pending investment by an OPEC member
state in any "downstream" operation.

<smry> Summary:
Document must identify an existing or pending investment by an OPEC member
state in any "downstream" operation.

<narr> Narrative:
To be relevant, a document must identify an existing or pending investment by
an OPEC member state (or its national oil company) in any "downstream"
installation or enterprise.  The investment may be through a joint venture,
```



```
acquisition, construction, or stock purchase of any operation in the refining,
petrochemical, oil industry equipment manufacture, drilling and exploration,
shipping, marketing and retail sales, or other ancillary "downstream" activity.

<con> Concept(s):

1.  OPEC, Organization of Petroleum Exporting Countries

2.  Algeria, Ecuador, Gabon, Indonesia, Iran, Iraq, Kuwait,
    Libya, Nigeria, Qatar, Saudi Arabia, United Arab Emirates (UAE),
    Venezuela.

3.  Kuwait Petroleum Co., Kuwait Investment Co.

4.  Petroleos de Venezuela

<fac> Factor(s):

<def> Definition(s):

1.  "Downstream:" any petroleum industry activity which occurs after initial
    production of crude oil, or which supports such production.

</top>
```


Beispiel für ein „Topic“aus TREC-2:

```
<top>

<head> Tipster Topic Description

<num> Number: 123

<dom> Domain: Medical & Biological

<title> Topic: Research into & Control of Carcinogens

<desc> Description:

Document will report on studies into linkages between environmental factors or
chemicals which might cause cancer, and/or it will report on governmental
actions to identify, control, or limit exposure to those factors or chemicals
which have been shown to be carcinogenic.

<smry> Summary:

Document will report on studies into linkages between environmental factors or
chemicals which might cause cancer, and/or it will report on governmental
actions to identify, control, or limit exposure to those factors or chemicals
```



```
which have been shown to be carcinogenic.

<narr> Narrative:

A relevant document will report on research into linkages between cancer and
environmental hazards and/or the efforts of governments to limit exposure of
their people to carcinogens.  The governmental action may be of any category,
e.g. entry into international agreements, enactment of domestic laws, issuance
of administrative regulations, support of carcinogen research, air and soil
sampling, launching of public education campaigns, etc.

<con> Concept(s):

1. cancer, carcinogen

2. treaty, agreement, law, regulation, study, research, education, Super Fund

<fac> Factor(s):

<def> Definition(s):

</top>
```


Beispiel für ein „Topic“aus TREC-3:

```
<top>

<num> Number:  180

<title> Topic: Ineffectiveness of U.S. Embargoes/Sanctions

<desc> Description:

Document will report ineffective results of embargo/sanction
activities against a foreign nation by the U.S.  Embargo may
be established in conjunction with friendly nation(s) after
notification to the U.N.

<narr>  Narrative:

A relevant document will provide statement or implication on the
ineffectiveness of an embargo or sanction taken against a foreign
nation by the U.S.  Document must also contain the type and reason for
the embargo or sanction, name of foreign nation, and, where noted,
information regarding the method(s) employed by that foreign nation
to circumvent the embargo or sanction.

</top>
```


Beispiel für ein „Topic“aus TREC-4:

```
<top>

<num> Number:  227

<desc> Description:

Identify instances and reasons of deaths in the U.S.
military caused by other than enemy (e.g., friendly fire,
training accidents).

</top>
```


Beispiel für ein „Topic“aus TREC-5:

```
<top>

<num> Number: 283
<title> Topic: China Trade

<desc> Description:
Has the China trade policy had a positive effect on
U.S. consumers?

<narr> Narrative:
Chinese manufacturers' exploitation of the labor force
and poor working conditions have brought criticism from
the U.S. and some European organizations, particularly
organized labor unions.  They contend that U.S. consumers
are being offered products inferior in both quality of
material and workmanship.  U.S. companies allege that,
with using foreign manufacturers, consumer costs remain
relatively stable with minimal degradation in quality and
workmanship of the product involved.

</top>
```


Beispiel für ein „Topic“aus TREC-6:

```
<top>

<num> Number: 329
<title> Mexican Air Pollution

<desc> Description:
Mexico City has the worst air pollution in the
world.  Pertinent Documents would contain the
specific steps Mexican authorities have taken
to combat this deplorable situation.

<narr> Narrative:
Relevant documents would discuss the steps the
Mexican Government has taken to alleviate the
air pollution in Mexico City.  Steps such as
reducing the number of automobiles in the city,
encouraging the use of mass public transportation,
and creating new mass transportation systems are
relevant, among others.  Mention of any new
methods in the design stage would also be
appropriate.

</top>
```


Beispiel für ein „Topic“aus TREC-7:

```
<top>

<num> Number: 352
<title> British Chunnel impact

<desc> Description:
What impact has the Chunnel had on the British economy and/or
the life style of the British?

<narr> Narrative:
Documents discussing the following issues are relevant:

 - projected and actual impact on the life styles of the British
 - Long term changes to economic policy and relations
 - major changes to other transportation systems linked with the Continent

Documents discussing the following issues are not relevant:

 - expense and construction schedule
 - routine marketing ploys by other channel crossers (i.e.,
   schedule changes, price drops, etc.)

</top>
```


Ermittlung der relevanten Dokumente

Bei TREC werden die zu einem *topic* relevanten Dokumente aus einem *pool* von möglicherweise relevanten Dokumenten ermittelt.

Der *pool* wird dadurch erzeugt, dass man über alle Einreichungen die Vereinigung über deren K relevanteste Dokumente bildet (typisch $K = 100$).

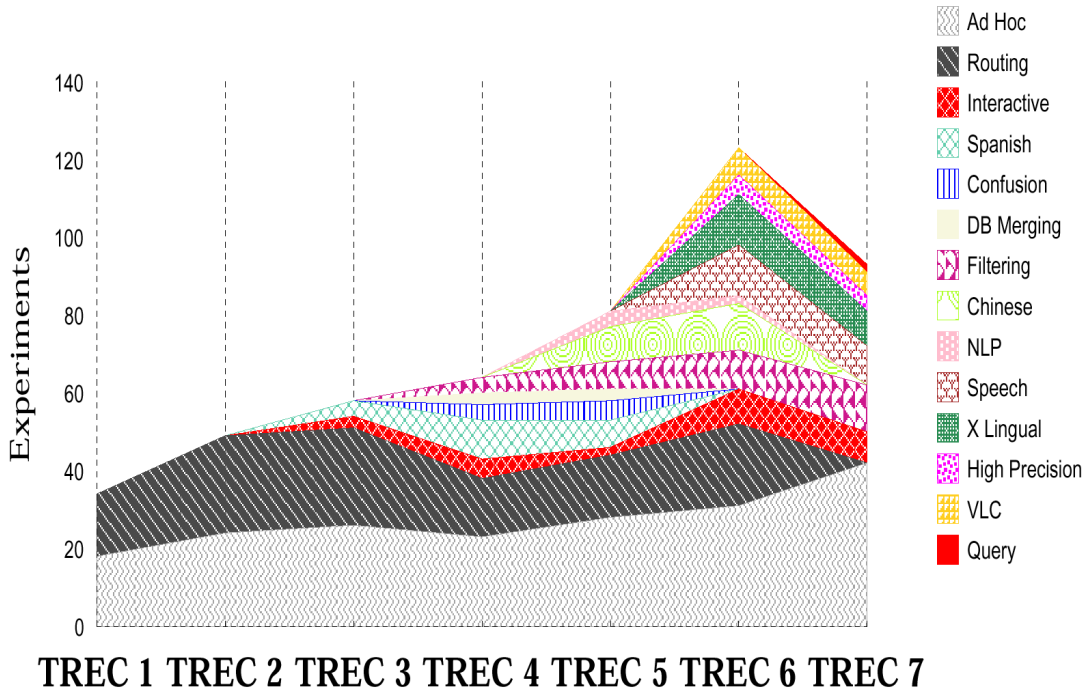
Die Dokumente aus dem *pool* werden dann menschlichen Experten zur Beurteilung vorgelegt.

Diese *pooling method* basiert auf folgender Annahme:

- (fast) alle relevanten Dokumente werden im *pool* enthalten sein,
- die Dokumente, die nicht im *pool* sind, sind daher auch nicht relevant.

Zur Belegung dieser Annahmen wurden im Rahmen von TREC weitreichende Untersuchungen gemacht.

Anzahl der TREC-Experimente nach Art der bearbeiteten Aufgabe [VH98]



Aufgaben (*tasks*) in den verschiedenen Teilbereichen (*tracks*)

- *Ad hoc Tasks*

  Hier wird die Suche in einer statischen Dokumentenkollektion betrachtet.

  Man kann also mit der Dokumentenkollektion beliebig viel „üben“.

  Man bekommt dann aber einige *neue* Anfragen, die auf der Basis des geübten beantwortet werden müssen.

  In TREC-7 bestand die Datenkollektion aus ca. 2 GB mit Dokumenten (CDs 4 und 5) und einer Menge von 50 natürlichsprachlichen Anfragen (*topics*).

  Die Teilnehmer erstellen für diese *topics* eine Menge von Anfragen, die von ihnen auf die Datenbasis angewendet werden.

  Technisch läuft dies wie folgt ab:

  – TREC gibt eine Dokumentenkollektion und die Anfragen aus.

  – Die von einer Expertengruppe erarbeiteten „objektiven“ gibt es zu diesem Zeitpunkt aber noch nicht.

- Man kann das System nun anhand der Kollektionen vergangener TREC-Konferenzen (mit bekannten Relevanzurteilen) oder anhand anderer Kollektionen optimieren.
- Die Ergebnisse zu den neuen *topics* sind von den Teilnehmern einzureichen.
- Die Ergebnisse werden mit den Relevanzurteilen der Experten zu diesen 50 Anfragen verglichen.
- Die Teilnehmer geben dazu je *topic* ein Ranking der von ihrem System ermittelten 1000 relevantesten Dokumente ab.

Welche Verfahren eingesetzt werden, ist dabei vollkommen frei.

TREC unterscheidet lediglich vollautomatische und manuelle Verfahren.

Bei den vollautomatischen Verfahren wird die Anfrage aus der *topic*-Beschreibung ohne manuelle Interaktion gewonnen.

- *Routing task*

  Aufgabe: durch feststehende Anfragen Informationen aus einem Nachrichtenstrom zu bestimmten Nachfragern routen.

  In TREC müssen bekannte *topics* aus vorhergehenden TREC-Konferenzen (d.h. auch die Relevanzurteile sind für die alten Dokumentkollektionen bekannt) in Anfragen umgesetzt werden, die dann auf bisher unbekannte Dokumente angesetzt werden.

  Dabei wird aber ein Ranking geliefert!

  Diese Aufgabe wurde ab TREC-7 durch die *Filtering Task* ersetzt.

- *Interactive Task*

  Hier geht es um Systeme, die bei der Anfrage eine Benutzerinteraktion benötigen.

  Z.B. der Benutzer markiert in einem ersten Ergebnis welche Dokumente relevant sind, das System lernt dann daraus für eine weitere Runde.

  Versuchsaufbau: Ein Testnutzer muß in 15 Minuten mit dem System so viele relevante Dokumente wie möglich finden.

- *Spanish Task*

  Während die übrigen Aufgaben in Englisch ablaufen, wird hier Spanisch verwendet.

  Ansonsten gleicht die Aufgabe den *Ad hoc Tasks*.

- *Confusion Task*

  Hier werden Dokumente gesucht, die durch Surrogate repräsentiert werden. Ein Beispiel sind gesprochene Texte, ein anderes Texte, die per OCR eingelesen werden.

  Diese Task ist seit TREC-7 durch die Spoken Document Retrieval (SDR) Task ersetzt.

- *DB Merging task*

  Ziel: Ergebnisse, die bei mehreren Suchen (in verschiedenen Kollektionen) ermittelt wurden, sollen zu einem Gesamtergebnis „gemischt“ werden.

- *Filtering task*

  Hier ist die Aufgabenstellung im Prinzip wie beim Routing.

  Es wird jedoch eine binäre Entscheidung gefordert.

- *Chinese*

  Während die übrigen Aufgaben in Englisch ablaufen, wird hier Chinesisch verwendet.

  Ansonsten gleicht die Aufgabe den *Ad hoc Tasks*.

- *Natural Language Processing (NLP)*

  Mit dieser Task soll untersucht werden, ob Systeme, die Verfahren des Sprachverstehens einsetzen, effizienter arbeiten als „traditionelle“ IR-Systeme.

- *Spoken Document Retrieval (SDR) Task (Speech)*

  In dieser Task wird eine Dokumentmenge aus transscribierten Radioaufzeichnungen verwendet.

  Dazu muss man auf eine Lautschrift zurückgreifen, wodurch sich Ungenauigkeiten und Mehrdeutigkeiten ergeben.

- *Cross Language (CLIR) task (X Lingual))*

  Anfragen zu einer Dokumentenkollektion in der Sprache X werden in der Sprache Y gestellt. $Y \neq X$

  Beispiel: X = Englisch, Y = Deutsch

  Sinn: Anfrager stellt die Anfrage in seiner Muttersprache, bekommt aber auch Dokumente in anderen Sprachen nachgewiesen.

- *High Precision Task*

  Die Aufgabe besteht hier darin, innerhalb von 5 Minuten 15 hochgradig relevante Dokumente zu finden.

- *Very Large Corpus (VLC) Task*

  Hier wird das Verhalten von IR-Systemen auf sehr großen Dokumentmengen untersucht.

  Bei TREC-7 waren es 100 GB.

- *Query Task*

  Das Ziel ist hier eine große Menge von Anfragen zu einem *topic* zu erstellen.

  Der Hintergedanke ist dabei anfragespezifische Bearbeitungsstrategien zu entwickeln.

  Man unterscheidet:

  – Anfragen mit 2 bis 3 Wörten,
  – Anfragen aus einem Satz,
  – Anfragen aus einem Satz, die durch ein manuelles Feedback auf 5 bis 10 gefundene Dokumente verbessert werden,
  – ...

Kriterien zum Vergleich der Systeme

P(10): Die Precision, nachdem die ersten 10 Dokumente gefunden wurden.

P(30): Die Precision, nachdem die ersten 30 Dokumente gefunden wurden.

R-Prec: Die Precision, nachdem die ersten R Dokumente gefunden wurden, R ist dabei die Anzahl der zu dem *topic* relevanten Dokumente.

Mean Ave Precision: Die durchschnittliche Precision.

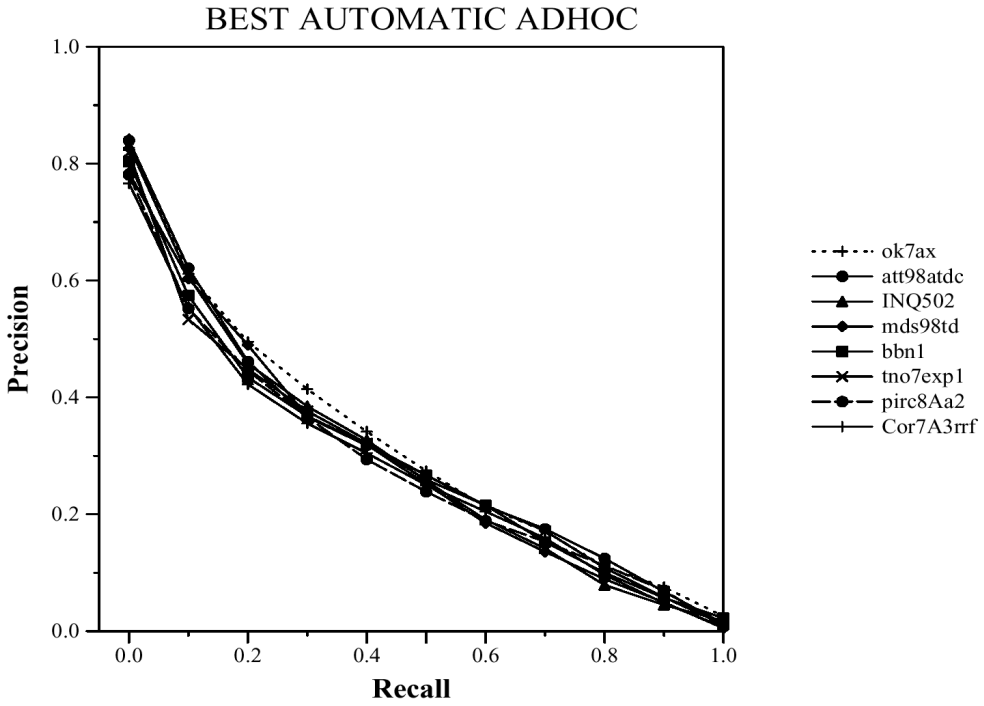
Recall at .5 Prec: Der Recall in dem Rang, in dem die Precision zum ersten mal unter 50% fällt (und min. 10 Dokumente geliefert sind).

R(1000): Der Recall, nachdem 1000 Dokumente ermittelt wurden.

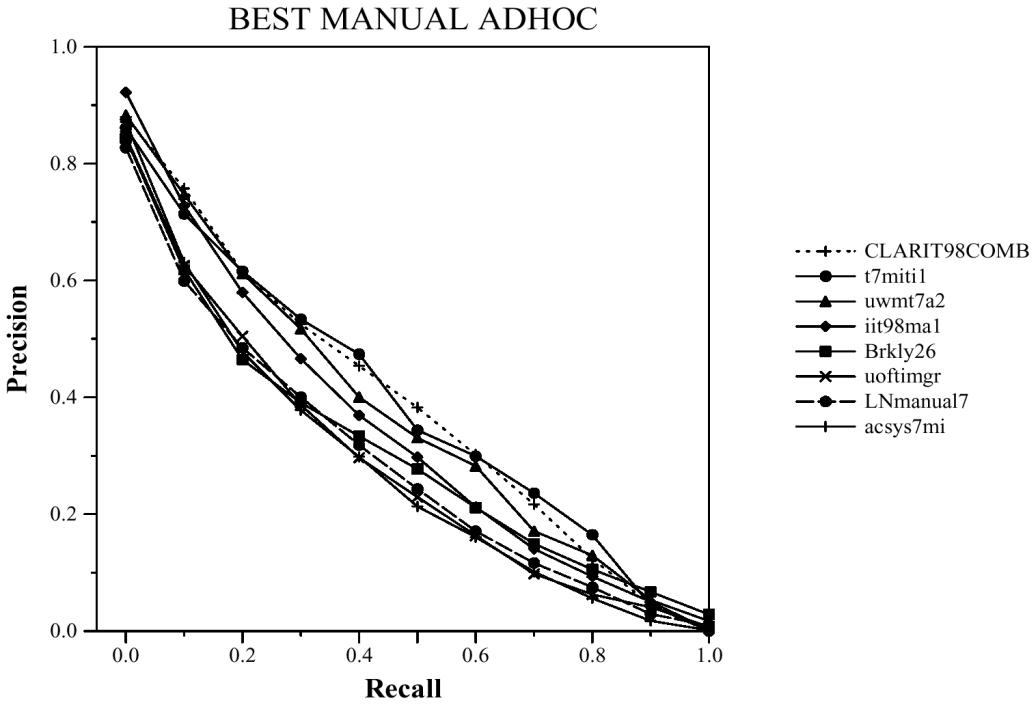
Total Rel Ret: Die Summe der gelieferten relevanten Dokumente über alle 50 *topics*

Rank 1st Rel: Der Rang, in dem das erste relevante Dokument ermittelt wurde.

Recall/Precision-Graph der 8 besten autom. Systeme bei TREC-7



Recall/Precision-Graph der 8 besten manuellen Systeme bei TREC-7



2.4.2 Die CACM- und CISI-Kollektionen

Ein Nachteil der TREC-Kollektion ist ihr Umfang.

Durch den Umfang sind die Anforderungen an die Hardware recht hoch und auch die Vorbereitungen für Messungen und die Messungen selbst sind sehr aufwendig.

Daher besteht auch ein Bedarf für kleinere Testkollektionen.

Zusätzlich können solche Kollektionen Eigenschaften haben, die in den TREC-Kollektionen nicht enthalten sind (z.B. strukturierte Dokumente).

Hierzu werden häufig z.B. die folgenden Kollektionen verwendet:

- ADI (82 Dokumente aus dem Bibliothekswesen),
- CACM (3204 Artikel aus den Communications of the ACM),
- INSPEC (12684 abstracts on electronics, computer, and physies),
- CISI, (1460 Dokumente aus dem Bibliothekswesen) und
- Medlars (medizinische Artikel).

Die CACM-Kollektion

Es handelt sich um 3204 Artikel, die in den *Communications of the ACM* zwischen 1958 und 1979 veröffentlicht wurden.

Damit wird ein beträchtlicher Teil der Informatik-Literatur thematisch abgedeckt, da die CACM damals — und immer noch — eine der Top-Zeitschriften in der Informatik ist.

Neben den Dokumenten selbst enthalten die Datensätze noch weitere Informationen:

- die Namen der Autoren
- das Datum des Erscheinens
- Wortstämme aus der Überschrift und den Abstracts
- Kategorien, die aus der ACM Computing Reviews Klasifikation abgeleitet sind
- direkte Querverweise zwischen Artikeln (Paare mit der Bedeutung a referenziert b)
- bibliographische gemeinsame Beziehungen (Tripel (a, b, c)) mit der Bedeutung a und b enthalten eine Referenz auf c)
- Anzahl gemeinsamer Zitate für jedes Paar von Dokumenten

Damit eignet sich diese Kollektion insbesondere zum Test von Verfahren, die Zitate mit betrachten.

Die CACM-Kollektion umfaßt auch 52 Beispielanfragen. Beispiel:

What articles exist which deal with TSS (Time Sharing System), an operating system for IBM computers?

Zu jeder Anfrage existieren auch 2 boolesche Anfrageformulierungen und die Menge der relevanten Dokumente.

Da die Anfragen sehr spezifisch sind, gibt es im Durchschnitt ca. 15 relevante Dokumente.

Ein Beispieldokument aus der CACM-Kollektion

```
.I 3198
.T
   Microprogramming, Emulators and Programming Languages
.W
  The problem we have been concerned with is that of converting language to
action - or intellectual energy to mechanical energy. The medium that we use
for this purpose is language and therefore we are preoccupied with the subject
of language. In the areas of language investigation we have concentrated first
on formalizing syntax and then on semantics.
.B
CACM March, 1966
.A
Greem, J.
.N
CA660318 ES March 17, 1982 10:10 AM
.X
1542    5       3198
3198    5       3198
3198    5       3198
3198    5       3198
1491    6       3198
3198    6       3198
```


Die CISI-Kollektion

Diese Kollektion enthält 1460 Dokumente und wurde aus einer Vorläuferkollektion abgeleitet, die am *Institute of Scientific Information* (CISI) entwickelt wurde.

Die Kollektion kann hauptsächlich zur Evaluierung von Verfahren genutzt werden, die auf Ähnlichkeiten und Kreuzzitaten aufbauen.

Auch in dieser Kollektion werden zusätzliche Informationen bereitgestellt:

- die Namen der Autoren
- Wortstämme aus der Überschrift und den Abstracts
- Anzahl gemeinsamer Zitate für jedes Paar von Dokumenten

Es gibt 35 Beispielanfragen in natürlicher Sprache, zu denen aber auch boolesche Anfragen vorliegen.

Ferner gibt es weitere 41 Anfragen, die nur in natürlicher Sprache vorliegen.

Zu jeder Anfrage gibt es dabei im Durchschnitt 50 relevante Dokumente.

Dabei haben aber viele der relevanten Dokumente keine Begriffe mit der Anfrage gemein!

Ein Beispieldokument aus der CISI-Kollektion

```
.I 6
.T
Abstracting Concepts and Methods
.A
Borko, H.
.W
     Graduate library school study of abstracting should be more than a
how-to-do-it course.
It should include general material on the characteristcs and types of abstracts,
the historical development of abstracting publications, the abstract-publishing
industry (especially in the United States), and the need for standards in the
preparation and evaluation of the product.
These topics we call concepts.
     The text includes a methods section containing instructions for writing
various types of abstracts, and for editing and preparing abstracting publications.
These detailed instructions are supplemented by examples and exercises in the
appendix.
There is a brief discussion of indexing of abstract publications.
     Research on automation has been treated extensively in this work, for we
believe that the topic deserves greater emphasis than it has received in the
past.
Computer use is becoming increasingly important in all aspects of librarianship.
```



```
Much research effort has been expended on the preparation and evaluation of
computer-prepared abstracts and extracts.
Students, librarians, and abstractors will benefit from knowing about this
research and understanding how computer programs were researched to analyze text,
select key sentences, and prepare extracts and abstracts.
The benefits of this research are discussed.
    Abstracting is a key segment of the information industry.
Opportunities are available for both full-time professionals and part-time or
volunteer workers.
Many librarians find such activities pleasant and rewarding, for they know
they are contributing to the more effective use of stored information.
One chapter is devoted to career opportunities for abstractors.
.X
6       2       6
6       2       6
6       2       6
6       2       6
6       2       6
6       2       6
403     2       6
461     2       6
551     2       6
363     2       6
```


Statistiken zu CACM und CISI

Die Dokumente:

Collection	*Num. Docs*	*Num. Terms*	*Terms/Docs.*
CACM	3204	10446	40,1
CISI	1460	7392	104,9

Die Anfragen:

Collection	*Number Queries*	*Terms per Query*	*Relevants per Query*
CACM	52	11.4	15.3
CISI	35 & 76	8.1	49.8

Bemerkenswert ist dabei, daß für die CISI-Kollektion die Anzahl der relevanten Dokumente pro Anfrage deutlich höher ist (3,4 % zu 0,5 %).

2.4.3 Weitere Testkollektionen

Collection	*Subject*	*Num. Docs*	*Num. Queries*
ADI	Inforniation Science	82	35
CACM	Cornputer Seience	3200	64
CISI	Library Science	1460	76
CRAN	Aeronauties	1398	225
LISA	Library Seience	6004	35
MED	Medieine	1033	30
NLM	Medieine	3078	155
NPL	Elee. Engineering	11,429	100
TIME	General Articles	423	83

Jede dieser Kollektionen hat ihre Spezifika.

Da die Kollektionen bereits häufig benutzt wurden, kann auf ausgiebige Erfahrungen zurückgegriffen werden.

Kapitel 3

Berücksichtigung der Vagheit in Sprache

Das Problem:

- Dokumente und Anfragen in natürlicher Sprache müssen in Repräsentationen überführt werden.

Ein einfacher Ansatz:

Repräsentation für Dokumente und Anfragen jeweils:
Die Menge der in Ihnen vorkommenden Wörter.

Text:

„Wirtschaft und Gesellschaft befinden sich derzeit im größten Umbruch seit der Industrialisierung. Die Ursache hierfür liegt in der globalen Verfügbarkeit leistungsfähiger und zugleich kostengünstiger Informations- und Kommunikationstechnologien. Das Informationszeitalter wird Realität.“

↓

{ befinden, Das, der, derzeit, Die, Gesellschaft, globalen, größten, hierfür, im, in, Industrialisierung, Informations, Informationszeitalter, Kommunikationstechnologien, kostengünstiger, leistungsfähiger, liegt, Realität, seit, sich, Umbruch, und, Ursache, Verfügbarkeit, wird, Wirtschaft, zugleich }

Probleme:

- Wörter, die keinen Sinn tragen, (z.B. *und*) werden aufgenommen.

⇒ Stoppworteliminierung

- Wenn ich nach *Kommunikation* und *Technologie* suche, werde ich das Dokument nicht finden, weil nur *Kommunikationstechnologien* vorkommt.

⇒ Mehrwortgruppenidentifizierung

- Suche ich nach *Haus*, und im Text kommt nur *Häuser* vor, finde ich den Text nicht.

⇒ Stammform- oder Grundformreduzierung

- Kommt im Text *Fernsprecher* vor, und ich suche nach *Telefon*, so werde ich den Text nicht finden.

⇒ Verwaltung von Synonymen, ...

3.1 Stoppworteliminierung

Ziel: Terme, die nicht zur Semantik der Dokumente beitragen, nicht verwalten!

Effekte:

- Reduzierung des Speicherplatzbedarfs für die Repräsentationen
- Verbesserung der Performance der Matchingalgorithmen

Es gibt zwei verschiedene Ansätze:

- Verwaltung einer Stoppwortliste
  Beispiel: *der, die, das, und, sowieso, ...*
  Verwaltung in einer effizienten internen Datenstruktur:
  - z.B. Hashtabelle, oder AVL-Baum
  - Einladen beim Start des IR-Systems
  - Alternative: externe Datenstruktur; z.B. B-Baum
- Eliminierung aller „hochfrequenten Begriffe“
  z.B. Eliminierung aller Begriffe, die in mehr als 20 % der Dokumente vorkommen

Eliminierung hoch- und niedrigfrequenter Begriffe

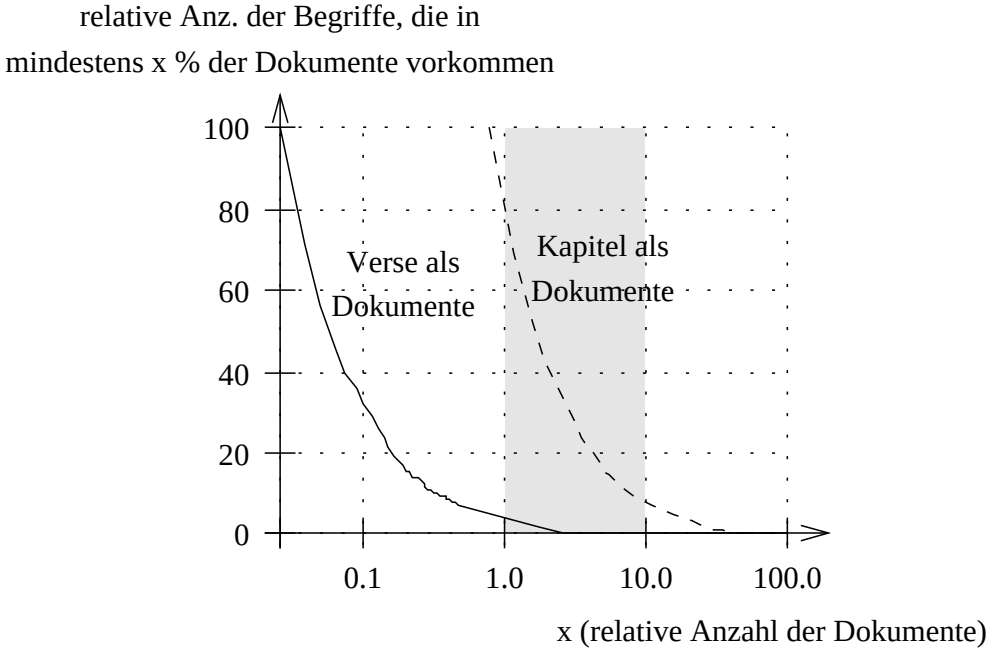
Zur Eliminierung hoch- und niedrigfrequenter Begriffe schlägt Crouch in [Cro90] vor,

- Begriffe, die in weniger als 1 % aller Dokumente auftreten, nicht zu betrachten, weil sie zu spezifisch sind, und
- Begriffe, die in mehr als 10 % der Dokumente auftreten, nicht zu betrachten, weil sie zu allgemein sind.

Dabei darf aber der **Einfluß der Größe der Dokumente** nicht unterschätzt werden.

Beispiel: *Kapitel und Verse des Neuen Testaments der Bibel*

- Wenn man die Verse als Dokumente betrachtet und nur Substantive berücksichtigt, dann enthält jedes Dokument im Durchschnitt nur 4,09 Begriffe.
- Betrachtet man Kapitel als Dokumente, so enthält jedes Dokument im Durchschnitt 76,34 Begriffe.



Für die **Kapitel** erscheinen die von Crouch angegebenen Grenzen relativ sinnvoll.

18 % der Begriffe würden danach als zu speziell und 8 % als zu allgemein eingestuft.

Bei den **Versen** würden dagegen über 96 % der Begriffe als zu speziell eingestuft.

Eine typische Stoppwortliste im Englischen (`ftp://ftp.cs.cornell.edu/pub/smart/english.stop`):

a	a's	able	about	above	according	accordingly	
across	actually	after	afterwards	again	against	ain't	all
allow	allows	almost	alone	along	already	also	although
always	am	among	amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway	anyways	anywhere	apart
appear	appreciate	appropriate	are	aren't	around	as	aside
ask	asking	associated	at	available	away	awfully	b
be	became	because	become	becomes	becoming	been	before
beforehand	behind	being	believe	below	beside	besides	best
better	between	beyond	both	brief	but	by	c
c'mon	c's	came	can	can't	cannot	cant	cause
causes	certain	certainly	changes	clearly	co	com	come
comes	concerning	consequently	consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently	d	definitely	described
despite	did	didn't	different	do	does	doesn't	doing
don't	done	down	downwards	during	e	each	edu
eg	eight	either	else	elsewhere	enough	entirely	especially
et	etc	even	ever	every	everybody	everyone	everything
everywhere	ex	exactly	example	except	f	far	few
fifth	first	five	followed	following	follows	for	former
formerly	forth	four	from	further	furthermore	g	get
gets	getting	given	gives	go	goes	going	gone
got	gotten	greetings	h	had	hadn't	happens	hardly
has	hasn't	have	haven't	having	he	he's	hello
help	hence	her	here	here's	hereafter	hereby	herein

hereupon	hers	herself	hi	him	himself	his	hither
hopefully	how	howbeit	however	i	i'd	i'll	i'm
i've	ie	if	ignored	immediate	in	inasmuch	inc
indeed	indicate	indicated	indicates	inner	insofar	instead	into
inward	is	isn't	it	it'd	it'll	it's	its
itself	j	just	k	keep	keeps	kept	know
knows	known	l	last	lately	later	latter	latterly
least	less	lest	let	let's	like	liked	likely
little	look	looking	looks	ltd	m	mainly	many
may	maybe	me	mean	meanwhile	merely	might	more
moreover	most	mostly	much	must	my	myself	n
name	namely	nd	near	nearly	necessary	need	needs
neither	never	nevertheless	new	next	nine	no	nobody
non	none	noone	nor	normally	not	nothing	novel
now	nowhere	o	obviously	of	off	often	oh
ok	okay	old	on	once	one	ones	only
onto	or	other	others	otherwise	ought	our	ours
ourselves	out	outside	over	overall	own	p	particular
particularly	per	perhaps	placed	please	plus	possible	presumably
probably	provides	q	que	quite	qv	r	rather
rd	re	really	reasonably	regarding	regardless	regards	relatively
respectively	right	s	said	same	saw	say	saying
says	second	secondly	see	seeing	seem	seemed	seeming
seems	seen	self	selves	sensible	sent	serious	seriously
seven	several	shall	she	should	shouldn't	since	six
so	some	somebody	somehow	someone	something	sometime	sometimes
somewhat	somewhere	soon	sorry	specified	specify	specifying	still

sub	such	sup	sure	t	t's	take	taken
tell	tends	th	than	thank	thanks	thanx	
that	that's	thats	the	their	theirs	them	themselves
then	thence	there	there's	thereafter	thereby	therefore	therein
theres	thereupon	these	they	they'd	they'll	they're	they've
think	third	this	thorough	thoroughly	those	though	three
through	throughout	thru	thus	to	together	too	took
toward	towards	tried	tries	truly	try	trying	twice
two	u	un	under	unfortunately	unless	unlikely	until
unto	up	upon	us	use	used	useful	uses
using	usually	uucp	v	value	various	very	via
viz	vs	w	want	wants	was	wasn't	way
we	we'd	we'll	we're	we've	welcome	well	went
were	weren't	what	what's	whatever	when	whence	whenever
where	where's	whereafter	whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who	who's	whoever	whole
whom	whose	why	will	willing	wish	with	within
without	won't	wonder	would	would	wouldn't	x	y
yes	yet	you	you'd	you'll	you're	you've	your
yours	yourself	yourselves	z	zero			

3.2 Stamm- und/oder Grundformreduktion

Problem: Wörter tauchen in verschiedenen **Flexionsformen** (*Beugungsformen*) auf:

- Konjugation: Beugung des Zeitwortes

  laufen — lief — gelaufen

- Deklination: Beugung des Haupt-, Eigenschafts-, Für- und Zahlwortes

  das Haus — des Hauses — die Häuser

Daneben wird der gleiche Wortstamm in verschiedenen Derivationsformen verwendet:

proben — Erprobung — Probe

Problem: ein Wort taucht in der Anfrage in einer bestimmten Form auf:

Um „alle" potentiell relevanten Dokumente mit einer Anfrage zu finden, sollte sich das Matching nicht auf die einzelnen Wortformen, sondern auf die Wortstämme beziehen!

Zwei grundsätzliche Ansätze

1. *Erweiterung der Anfrage*
   - In der Repräsentation bleiben sämtliche Wortformen erhalten
   - Die Anfrage wird um alle denkbaren Wortformen erweitert
   - Vorteile:
     – der Informationsgehalt der Repräsentationen ist vollständig
     – ggf. können auch Anfragen bearbeitet weden, die sich auf spezielle Wortformen beziehen
   - Nachteile:
     – die Repräsentationen werden sehr groß
     – die Performance des Systems wird problematisch

   ⇒ die Nachteile überwiegen i. Allg.

2. *Generelle Abbildung der Terme auf die Grund- oder Stammform*
   - Bei der Umwandlung von Anfrage und Dokumenten in die jeweilige Repräsentation werden die Wortformen reduziert
   - Vorteile:
     – die Zahl der zu verwaltenden Terme wird deutlich reduziert
     – die Performance der Algorithmen steigt
     – die Rückführung auf die Grund- oder Stammform ist etwas einfacher als eine Expansion auf alle Wortformen
   - Nachteile:
     – Eine gezielte Suche nach einer Wortform ist aufgrund der Repräsentation nicht mehr möglich
     – man verliert damit bei manchen Anfragen an *Precision*

   ⇒ für den praktischen Einsatz überwiegen i. Allg. die Vorteile

3.2.1 Begriffe

Grundformreduktion

- die Wörter werden auf ihre grammatikalische Grundform zurückgeführt
- Substantive auf den Nominativ singular
- Verben auf den Infinitiv
- geschieht i. Allg. durch Abtrennen der Flexionsendung und anschließende Rekodierung
- Beispiel: *applies* $\rightarrow$ *appl* $\rightarrow$ *apply*

Stammformreduktion

- die Wortformen werden auf ihren Stamm zurückgeführt
- es handelt sich dabei i. Allg. nicht um eine tatsächlich als Wort vorkommende Form
- der Stamm kann (und sollte) ggf. für Verb und Substantiv gleich sein
- Beispiel: *computer, compute, computation, computerization* $\rightarrow$ *comput*

Bezeichnungen nach Kuhlen [Kuh77]:

- *lexikographische Grundform*
  die Form, in der das Wort in einem Wörterbuch zu finden ist.
  Die durch Flexion möglicherweise entstandenen graphematischen Veränderungen der Grundformen werden rückgängig gemacht, d.h. die Wörter werden deflektiert und anschließend rekodiert.
- *formale Grundform*
  „... Wortfragmente, bei denen die „normalen“ englischen und fremdsprachigen (hauptsächlich lateinischen) Flexionsendungen abgetrennt werden, ohne daß die entstandenen Wortfragmente rekodiert würden.“
- *Stammform*
  nach linguistischen Prinzipien die Zeichenketten, die durch Deflexion und Abtrennen von Derivationsendungen entstehen.
  Diese Zeichenketten sollen soweit wie möglich durch Rekodierung vereinheitlicht werden.

Die verschiedenen Reduktionsformen nach Kuhlen am Beispiel (aus [Kuh77]):

Formale Grundform	*Textwörter*	*Lexikalische Grundform*	*Stamm-form*
absorb	absorb	absorb	absorb
⋮	absorbed	⋮	⋮
⋮	absorbing	⋮	⋮
⋮	absorbs	⋮	⋮
⋮	absorber	absorber	⋮
⋮	absorbers	⋮	⋮
absorbab	absorbable	absorbable	⋮
⋮	absorbably	⋮	⋮
absorbanc	absorbance	absorbance	⋮
⋮	absorbances	⋮	⋮
⋮	absorbancy	absorbancy	⋮
⋮	absorbancies	⋮	⋮
absorbant	absorbant	absorbant	⋮
⋮	absorbants	⋮	⋮
⋮	absorbantly	⋮	⋮
absorbtion	absorbtion	absorbtion	⋮
⋮	absorbtions	⋮	⋮
absorbtiv	absorbtively	absorbtive	⋮
⋮	absorbtive	⋮	⋮

3.2.2 Überblick zu Verfahren zur Grund- und Stammformreduktion

- Verfahren auf der Basis einfacher **Trunkierung**
- Verfahren auf der Basis von **Regeln**
- Verfahren auf der Basis von **Wörterbüchern**

Welche Verfahren angewendet werden können, hängt stark von der Sprache ab:

- Das **Englische** ist „schwach flektiert“, und deshalb gut mit Regeln in den Griff zu bekommen.
- Das **Italienische** ist stärker flektiert, aber immer noch mit – allerdings deutlich mehr – Regeln handhabbar.
- Das **Deutsche** kann mit Regeln nicht sinnvoll abgedeckt werden. Man muß daher auf Wörterbücher ausweichen.

3.2.3 Verfahren auf der Basis einfacher Trunkierung

Beschreibung eines Wortstammes durch Trunkierung.

Trunkierung wird i. Allg. explizit in der Anfrage vorgenommen.

Beispiel: Metazeichen zur Bildung von regulären Ausdrücken in UNIX (nach [Gul88])

Bedeutung:	**Metazeichen:**
Beliebiges einzelnes Zeichen	`.` (Punkt)
Beliebige Zeichenkette (auch die leere)	`.*`
Beliebige Wiederholung des vorangestellten Zeichens (auch keine)	`*`
Beliebige Wiederholung des vorangestellten Zeichens (mindestens 1)	`+`
0 oder 1 Wiederholung des vorangestellten Zeichens	`?`
Eines der Zeichen aus ...	`[ ... ]`
Eines der Zeichen aus dem Bereich ...	`[ a-e ]`
Eines der Zeichen aus den Bereichen ...	`[ a-eh-x ]`
Alle Zeichen außer ...	`[ ^... ]`
Fluchtsymbol	`\`

Beispiel für eine Anfrage mit einfacher Trunkierung:

*H[aä]us.** AND *Finanz.**

unterstellt, daß sich die einzelnen Anfragepattern jeweils auf Wörter im Text beziehen:

Suche alle Dokumente, in denen mindestens ein Wort vorkommt, daß „H[aä]us.“ entspricht* **und** *ein Wort, daß „Finanz.*“ entspricht*

Beispiel für ein System: `grep`-Befehl in UNIX; bezieht sich auf Dateien

Nachteile:

- Anfrageformulierung sehr aufwendig für den Benutzer
- reguläre Ausdrücke können extrem aufwendig werden
- Syntaktische Regeln der Sprache werden nicht vom System unterstützt

3.2.4 Lovins-Algorithmus zur Grundformreduktion [Lov68]

Einer von vielen im Prinzip sehr ähnlichen regelbasierten Algorithmen.

Algorithmus arbeitet **zweistufig**:

1. das Abtrennen der Endung
2. ggf. die Transformation der verbliebenen Endung des Wortes

Das Resultat ist zwar nicht immer die beste mögliche Grundform.

Wichtig ist in diesen Fällen aber vor allem die **Konsistenz**:

Alle Wörter in der Anfrage und den Dokumenten werden nach dem gleichen Verfahren umgeformt!

Einige Endungen, die in Lovins Algorithmus am Anfang entfernt werden

Länge	Endung	Bedingung
11	alistically	B
	arizability	A
	izationally	B
10	antialness	A
	arisations	A
	arizations	A
	entialness	A
9	allically	C
	antaneous	A
4	able	A
	ably	A
	ages	B
	ally	B
3	ism	B
1	e	A

Die Endungen sind absteigend nach ihrer Länge sortiert.

Innerhalb einer Länge erfolgt die Sortierung alphabetisch.

Bedingungen:

A: keine Einschränkungen

B: verbleibender Wortstamm min. 3 Zeichen lang

C: verbleibender Wortstamm min. 4 Zeichen lang

1. Falls ein Wort mit einem Konsonanten $\neq s$, der von einem s gefolgt wird, endet:
   $\Rightarrow$ Entferne das s
   - *stems* $\rightarrow$ *stem*
   - aber *stress* $\rightarrow$ *stress*

2. Falls ein Wort mit *es* endet:
   $\Rightarrow$ Entferne das abschließende s
   - *places* $\rightarrow$ *place*
   - *likes* $\rightarrow$ *like*
   - *theses* $\rightarrow$ *these* („Fehler!“)
   - *indices* $\rightarrow$ *indice* („Fehler!“)
   - *syntheses* $\rightarrow$ *synthese* („Fehler!“)

   Offensichtlich führt diese Regel bei Wörtern griechischen Ursprungs, deren Singular mit *is* endet, zu Problemen.

   Man könnte versuchen, dies durch weitere Regeln in den Griff zu bekommen.

3. Überführe *iev* $\rightarrow$ *ief* und *metr* $\rightarrow$ *meter*.
   - *believable* $\rightarrow$ *believ* $\rightarrow$ *belief* (1. Schritt = Abschneiden der Endung)

4. Falls ein Wort mit *ing* endet:
   $\Rightarrow$ Lösche das *ing* es sei denn, das Wort besteht nach der Löschung nur aus einem Buchstaben oder aus *th*.
   - *thinking* $\rightarrow$ *think*
   - *singing* $\rightarrow$ *sing*
   - *sing* $\rightarrow$ *sing* (keine Änderung)
   - *thing* $\rightarrow$ *thing* (keine Änderung)
   - *preceding* $\rightarrow$ *preced* (Fehler; kein Wort!)

   Den Fehler im letzten Beispiel könnte man z.B. durch eine nachgeschaltete Regel beheben, die besagt:

   Sofern ein Wort nach der Reduktion mit *et*, *ed* oder *es* endet, wird ein *e* hinzugefügt.

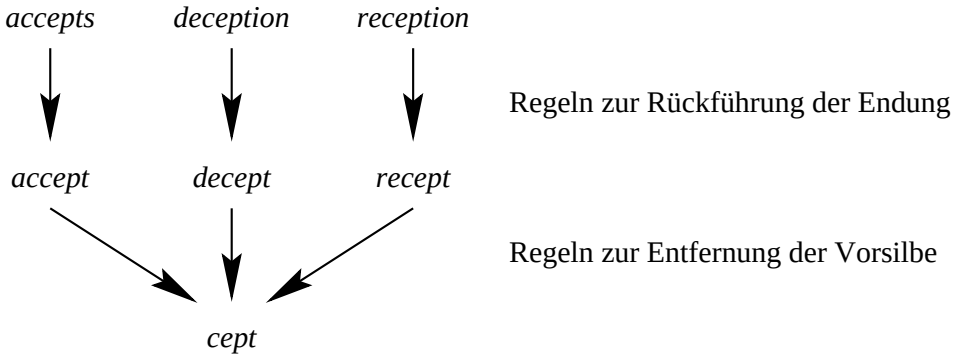
5. Falls ein Wort mit *ed* endet und ein Konsonant vorausgeht:
   ⇒ Lösche das *ed*, es sei denn, das Wort besteht nach der Löschung nur aus einem Buchstaben.
   - *ended* → *end*
   - *red* → *red*
   - *proceed* → *proceed* (es geht kein Konsonant voraus)
   - *proceeded* → *proceed*

6. Falls nach dem Entfernen der Endung ein Wort mit *bb*, *dd*, ..., *tt* endet:
   ⇒ Entferne einen der doppelten Buchstaben.
   - *embedded* → *embedd* → *embed*

7. Falls ein Wort mit *ion* endet:
   ⇒ Entferne *ion*, sofern das verbleibende Wort mehr als 2 Buchstaben hat.
   Sofern der letzte Buchstabe des Stammes ein Konsonant ist und der vorhergehende ein Vokal, füge zusätzlich ein *e* an.
   - *direction* → *direct*
   - *polution* → *polute*
   - *plantation* → *plantate* (Fehler!?)
   - *zion* → *zion*
   - *scion* → *scion*
   - *anion* → *anion*
   - *cation* → *cate* (Fehler!)

   Man beachte, daß *cation* (das Kation) ein Kunstwort aus *cathode* und *ion* ist. Folglich gibt es hier keinen eigentlichen Wortstamm.

- Ein arbeitsfähiges System benötigt wohl mindestens 10 bis 20 Regeln
- Ferner sind zahlreiche Ausnahmen z.B. für irreguläre Verben erforderlich.
- Das Programm sollte ferner die iterative Anwendung der Regeln unterstützen.
  z.B. *directions* → *direction* → *direct*
- Zusätzliche Programme zur Entfernung von Vorsilben sind zwar möglich, erscheinen aber für die Suche wenig sinnvoll.



  – Beispiel:
  - Obwohl alle drei Wörter vom gleichen lateinischen Stamm herrühren, sind die heutigen Bedeutungen doch recht unterschiedlich.
  - Folge: Man würde zu viel Unterscheidungskraft verlieren!

3.2.5 Porters-Algorithmus zur Grundformreduktion [Por80]

Zur Darstellung des Algorithmus benötigen wir einige Definitionen:

- Ein *Konsonant* in einem Wort ist ein Buchstabe ungleich A, E, I, 0 und U, und ungleich einem Y, dem ein *Konsonant* vorausgeht.
- So sind in TOY die Konsonanten T und Y enthalten,
- in SYZYGY sind es S, Z und G.
- Falls ein Buchstabe kein Konsonant ist, ist er ein *Vokal*.
- Einen *Konsonanten* werden wir im weiteren durch ein c bezeichnen und einen *Vokal* durch ein v.
- Eine Folge ccc... von mehr als 0 Konsonanten werden wir mit C bezeichnen und eine Folge vvv... von mehr als 0 Vokalen mit V.

Dies führt dazu, dass jedes Wort in einer der folgenden **4 Formen** dargestellt werden kann:

- CVCV...C
- CVCV...V
- VCVC...C
- VCVC...V

Wir können dafür zusammenfassend die Form [C]VCVC...[V] vewenden, wobei die eckigen Klammern für optionale Teile stehen.

Mit der Vereinfachung $(VC)^m$ geben wir an, dass VC m-mal wiederholt wird.

Damit können wir ein Wort wie folgt beschreiben: [C] $(VC)^m$ [V]

m werden wir als das Maß eines Wortes oder Wortteiles bezeichenen, der in dieser Form dargestellt wird.

Der Fall $m = 0$ entspricht einen „Nullwort“.

Einige *Beispiele*:

- $m = 0$ TR, EE, TREE, Y, BY.
- $m = 1$ TROUBLE, OATS, TREES, IVY.
- $m = 2$ TROUBLES, PRIVATE, OATEN, ORRERY.

Die Regeln zum Entfernen einer Endung werden wir in der folgenden Form angeben:

(Bedingung) S1 → S2

Dies bedeutet: Sofern ein Wort mit der Endung S1 endet und der Stamm vor S1 die Bedingung erfüllt, wird S1 durch S2 ersetzt.

Die Bedingung bezieht sich dabei im Allgemeinen auf m.

z.B.: $(m > 1)$ EMENT →

In diesem Fall ist S1 „EMENT“ und S2 das leere Wort.

Durch die obige Regel würde REPLACEMENT zu REPLAC, weil REPLAC ein Wortteil mit $m = 2$ ist.

Die Bedingung kann zusätzlich folgende Bestandteile enthalten:

- *S — der Stamm endet mit S (und analog für alle anderen Buchstaben).
- *v* — der Stamm enthält einen Vokal.
- *d — der Stamm endet mit einem doppelten Konsonanten (z.B. -TT, -SS).
- *o — der Stamm endet mit cvc, wobei das zweite c nicht für ein W, X oder Y steht (z.B. -WIL, -HOP).

Ferner können in einer Regel boolesche Operatoren angewendet werden.

In einer Folge von Regeln wird immer nur genau eine angewendet, und zwar die mit der längsten passenden Zeichenkette S1.

Der eigentliche Algorithmus

Schritt 1a

Die folgenden Regeln haben alle eine leere Bedingung; rechts werden Beispiele gegeben:

SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
			ties	→	ti
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Schritt 1b

$(m > 0)$	EED	→	EE	feed	→	feed
				agreed	→	agree
(*v*)	ED	→		plastered	→	plaster
				bled	→	bled
(*v*)	ING	→		motoring	→	motor
				sing	→	sing

Falls die 2. oder 3. Regel aus Schritt 1b zum Tragen kommt, wird zusätzlich folgende Verarbeitung durchgeführt:

AT	→	ATE	conflat(ed)	→	conflate
BL	→	BLE	troubl(ing)	→	trouble
IZ	→	IZE	siz(ed)	→	size
(*d and not (*L or *S or *Z))					
→ single letter			hopp(ing)	→	hop
			tann(ed)	→	tan
			fall(ing)	→	fall
			hiss(ing)	→	hiss
			fizz(cd)	→	fizz
($m = 1$ and *o)	→	E	fail(ing)	→	fail
			fil(ing)	→	file

Schritt 1c

(*v*)	Y	→	I	happy → happi
				sky → sky

Schritt 1 dient zusammengefaßt der Behandlung von Plural und Vergangenheitsformen.

Schritt 2

$(m > 0)$	ATIONAL	→	ATE	relational	→	relate
$(m > 0)$	TIONAL	→	TION	conditional	→	condition
				rational	→	rational
$(m > 0)$	ENCI	→	ENCE	valenci	→	valence
$(m > 0)$	ANCI	→	ANCE	hesitanci	→	hesitance
$(m > 0)$	IZER	→	IZE	digitizer	→	digitize
$(m > 0)$	ABLI	→	ABLE	conformabli	→	conformable
$(m > 0)$	ALLI	→	AL	radicalli	→	radical
$(m > 0)$	ENTLI	→	ENT	differentli	→	different
$(m > 0)$	ELI	→	E	vileli	→	vile
$(m > 0)$	OUSLI	→	OUS	analogousli	→	analogous
$(m > 0)$	IZATION	→	IZE	vietnamization	→	vietnamize
$(m > 0)$	ATION	→	ATE	predication	→	predicate
$(m > 0)$	ATOR	→	ATE	operator	→	operate
$(m > 0)$	ALISM	→	AL	feudalism	→	feudal
$(m > 0)$	IVENESS	→	IVE	decisiveness	→	decisive
$(m > 0)$	FULNESS	→	FUL	hopefulness	→	hopeful
$(m > 0)$	OUSNESS	→	OUS	callousness	→	callous
$(m > 0)$	ALITI	→	AL	formaliti	→	formal
$(m > 0)$	IVITI	→	IVE	sensitiviti	→	sensitive
$(m > 0)$	BILITI	→	BLE	sensibiliti	→	sensible

Schritt 3

$(m > 0)$	ICATE	→	IC	triplicate	→	triplic
$(m > 0)$	ATIVE	→		formative	→	form
$(m > 0)$	ALIZE	→	AL	formalize	→	formal
$(m > 0)$	ICITI	→	IC	electriciti	→	electric
$(m > 0)$	ICAL	→	IC	electrical	→	electric
$(m > 0)$	FUL	→		hopeful	→	hope
$(m > 0)$	NESS	→		goodness	→	good

Schritt 4

$(m > 1)$	AL	$\rightarrow$		revival	$\rightarrow$	reviv
$(m > 1)$	ANCE	$\rightarrow$		allowance	$\rightarrow$	allow
$(m > 1)$	ENCE	$\rightarrow$		inference	$\rightarrow$	infer
$(m > 1)$	ER	$\rightarrow$		airliner	$\rightarrow$	airlin
$(m > 1)$	IC	$\rightarrow$		gyroscopic	$\rightarrow$	gyroscop
$(m > 1)$	ABLE	$\rightarrow$		adjustable	$\rightarrow$	adjust
$(m > 1)$	IBLE	$\rightarrow$		defensible	$\rightarrow$	defens
$(m > 1)$	ANT	$\rightarrow$		irritant	$\rightarrow$	irrit
$(m > 1)$	EMENT	$\rightarrow$		replacement	$\rightarrow$	replac
$(m > 1)$	MENT	$\rightarrow$		adjustment	$\rightarrow$	adjust
$(m > 1)$	ENT	$\rightarrow$		dependent	$\rightarrow$	depend
$(m > 1)$	and (*S or *T) ION	$\rightarrow$		adoption	$\rightarrow$	adopt
$(m > 1)$	OU	$\rightarrow$		homologou	$\rightarrow$	homolog
$(m > 1)$	ISM	$\rightarrow$		communism	$\rightarrow$	commun
$(m > 1)$	ATE	$\rightarrow$		activate	$\rightarrow$	activ
$(m > 1)$	ITI	$\rightarrow$		angulariti	$\rightarrow$	angular
$(m > 1)$	OUS	$\rightarrow$		homologous	$\rightarrow$	homolog
$(m > 1)$	IVE	$\rightarrow$		effective	$\rightarrow$	effect
$(m > 1)$	IZE	$\rightarrow$		bowdlerize	$\rightarrow$	bowdler

Nun sind die Endungen entfernt, und es bleiben nur ein paar „Aufräumungsarbeiten“.

Schritt 5a

$(m > 1)$ E	$\rightarrow$		probate	$\rightarrow$	probat
			rate	$\rightarrow$	rate
$(m = 1$ and not *o) E	$\rightarrow$		cease	$\rightarrow$	ceas

Schritt 5b

$(m > 1$ and *d and *L)	$\rightarrow$	single letter	controll	$\rightarrow$	control
			roll	$\rightarrow$	roll

Der Algorithmus ist stark darauf ausgelegt, eine Endung nur dann zu entfernen, wenn der verbleibende Stamm noch lang genug ist.

Die Länge des Stammes wird dabei durch m abgeschätzt.

Hierfür gibt es keine linguistische Begründung.

Vielmehr haben empirische Untersuchungen ergeben, dass die Nutzung von m recht gut funktioniert.

So wird in den beiden folgenden Listen -ATE in Liste B entfernt, nicht jedoch in Liste A:

Liste A	Liste B
RELATE	DERIVATE
PROBATE	ACTIVATE
CONFLATE	DEMONSTRATE
PIRATE	NECESSITATE
PRELATE	RENOVATE

Die Tatsache, dass nicht versucht wird, Vorsilben zu entfernen, führt leider manchmal zu etwas inkonsistent erscheinenden Ergebnissen.

So verliert PRELATE seine Endung nicht. ARCHPRELATE wird aber zu ARCHPREL.

In der Praxis spielt dies aber keine wesentliche Rolle, weil die Anwesenheit der Vorsilbe irrtümliche Abbildungen auf den gleichen Stamm eher unwahrscheinlich macht.

Komplizierte Endungen werden Schritt für Schritt aufgelöst.

Beispiel:

GENERALIZATIONS → GENERALIZATION (Schritt 1) → GENERALIZE (Schritt 2) → GENERAL (Schritt 3) → GENER (Schritt 4).

OSCILLATORS → OSCILLATOR (Schritt 1) → OSCILLATE (Schritt 2) → OSCILL (Schritt 4) → OSCIL (Schritt 5).

In einem Vokabular von 10000 Worten wurde folgender Effekt erreicht:

Anzahl der reduzierten Worte in Schritt	1:	3597
"	2:	766
"	3:	327
"	4:	2424
"	5:	1373
Anzahl der nicht reduzierten Worte:		3650

Es ergab sich insgesamt ein Vokabular aus 6370 verschiedenen Stämmen.

Porter [Por80] vergleicht seinen Algorithmus mit einem zuvor in Cambridge eingesetzten [Daw74] auf der Basis der Cranfield 200 Kollektion (CRAN):

earlier system		present system	
precision	recall	precision	recall
0	57.24	0	58.60
10	56.85	10	58.13
20	52.85	20	53.92
30	42.61	30	43.51
40	42.20	40	39.39
50	39.06	50	38.85
60	32.86	60	33.18
70	31.64	70	31.19
80	27.15	80	27.52
90	24.59	90	25.85
100	24.59	100	25.85

Der Punkt dabei ist, dass das vorherige viel aufwendigere System keine besseren Ergebnisse liefert.

Eigenschaften regelbasierter Grund- und Stammformreduktion

Man hat die Wahl:

- Man kann **sehr vorsichtig** vorgehen:
  Man formuliert nur wenige sehr restriktive Regeln; Folgen:
  - es bleiben viele Wörter übrig
  - z.B. beim Vektorraummodell hat man ein sehr umfangreiches Vokabular
  - der Recall wird im Allgemeinen eher schwach sein
  - die Precision wird dagegen eher hoch sein
- Man kann **sehr ambitioniert** vorgehen:
  Man formuliert viele und zum Teil eher grobe Regeln; Folgen:
  - es bleiben nur „wenige“ Wörter übrig
  - z.B. beim Vektorraummodell hat man ein eher kleines Vokabular
  - der Recall wird im Allgemeinen recht hoch sein
  - die Precision wird dagegen eher schwach sein

3.2.6 Verfahren, die auf Wörterbüchern basieren

Regelbasierte Verfahren sind für stark flektierte Sprachen ungeeignet!

Daher werden hier wörterbuchbasierte (lexikalische) Verfahren eingesetzt.

Ein entsprechendes **Wörterbuch sollte enthalten**:

- Flexionsform → Grundform
  - *lief* → *laufen*
  - *Häuser* → *Haus*
- Derivationsform → Grundform
  - *Lieblosigkeit* → *lieblos*
  - *Berechnung* → *rechnen*

Probleme:

- ständige **Pflege** erforderlich
- für eine konkrete Anwendung **Anpassung an Fachvokabular** nötig

3.3 Mehrwortgruppenidentifikation

Das Problem im Deutschen:

- zusammengesetzte Begriffe
- Wenn wir „Bundeskanzlerwahl“ suchen, wollen wir auch „die Wahl des Bundeskanzlers“ suchen. (Oder vielleicht auch nicht! z.B. im Restaurant: „die Wahl des Bundeskanzlers fiel auf Saumagen mit Kraut“!)
- Ein anderes Beispiel: „Straßenbahn“
  Hier wollen wir aber nicht finden „der Konkurrenzkampf zwischen Straße und Bahn . . . “

Tendenz:

- Wenn wir nur nach dem zusammengesetzten Wort suchen:
  Precision steigt! Recall sinkt!
- Wenn wir alle zusammengesetzten Wörter so weit wie möglich zerlegen:
  Precision sinkt! Recall steigt!

Das Problem im Englischen:

- es gibt (fast) keine zusammengesetzten Wörter
- oft bekommen Begriffe aber erst gemeinsam eine bestimmte Bedeutung
- Beispiele: „information retrieval“ oder „artificial inteligence“
- Wenn das IR System nur die Begriffe einzeln kennt, kann die Precision nicht besonders gut werden!

Tendenz:

- Im Englischen braucht man einen Ansatz um *compound terms* oder *phrases* zu finden!

3.3.1 Ein *natural language processing* Ansatz [Str94]

Das Problem ist im Englischen, dass man im IR versucht Texte durch die in Ihnen vorkommenden *Begriffe* zu charakterisieren.

Begriffe – oder wie man auch sagt *Konzepte* – werden dabei aber häufig nicht durch ein Wort, sondern durch eine *Wortgruppe* beschrieben.

Insbesondere sind die durch eine Wortgruppe beschriebenen *Konzepte* oft wesentlich „trennschärfer“ als die nur durch ein Wort beschriebenen *Konzepte*.

So kann z.B. der Begriff *joint venture* bei der Suche im Wall Street Jounal (WSJ) sehr wichtig sein, während die Komponenten *joint* und *venture* recht unspezifisch sind.

So könnten *joint* und *venture* in einer großen Datenbank wegen ihrer großen Vorkommenshäufigkeit sogar zu Stoppwörtern erklärt werden.

Dadurch wird in großen Dokuementenkollektionen die Berücksichtigung von Mehrwortgruppen (*phrasal terms*) sehr wichtig.

Als Beispiel betrachten wir Topic 104 aus der TREC-Kollektion:


```
<top>
<head> Tipster Topic Description
<num> Number: 104
<dom> Domain: Law and Government
<title> Topic: Catastrophic Health Insurance
<desc> Description:
Document will enumerate provisions of the U.S. Catastrophic Health Insurance
Act of 1988, or the political/legal fallout from that legislation.
<smry> Summary:
Document will enumerate provisions of the U.S. Catastrophic Health Insurance
Act of 1988, or the political/legal fallout from that legislation.
<narr> Narrative:
A relevant document will detail the content of the U.S. medicare act of 1988
which extended catastrophic illness benefits to the elderly, with particular
attention to the financing scheme which led to a firestorm of protest and a
Congressional retreat, or a relevant document will detail the political/legal
consequences of the catastrophic health insurance imbroglio and subsequent
efforts by Congress to provide similar coverages through a less-controversial
mechanism.
<con> Concept(s):
1. Catastrophic Coverage Act of 1988, Medicare Part B, Health Care Financing
Administration
2. catastrophic-health program, catastrophic illness, catastrophic care,
acute care, long-term nursing home care
3. American Association of Retired Persons, AARP, senior citizen, National
Committee to Preserve Social Security and Medicare
<fac> Factor(s):
<nat> Nationality: U.S.
</fac>
<def> Definition(s):
</top>
```


Betrachtung der relevanten Dokumente in den 100 am höchsten gerankten Dokumenten:

QUERY:104; NO. RELEVANT:21

REL DOCUMENT	RANK (no phrases)	RANK (phrases)
WSJ890918-0173	2	5
WSJ891004-0119	7	1
WSJ870723-0064	8	8
WSJ870213-0053	10	12
WSJS80608-0121	14	7
WSJ891005-0005	15	4
WSJ891009-0009	35	18
WSJ890920-0115	39	26
WSJ890928-0184	40	61
WSJ880609-0061	53	50
WSJ891009-0188	73	46
WSJ880705-0194	97	95
WSJ870601-0075	—	52
WSJ891005-0001	—	72
WSJ871028-0059	—	93

Mit Mehrwortgruppen werden z.B. *catastrophic-health program*, *acute care*, *home care*, and *senior citizen* berücksichtigt.

Eine Anfrage, die sich nur auf die Felder `<title>`, `<desc>` und `<narr>`bezieht wird weniger effektiv sein, als eine, die `<con>` einbezieht.

Insbesondere, ohne die Mehrwortgruppen, denn der *narrative* Abschnitt enthält weit weniger spezifische Begriffe.

So zeigen Broglio und Croft [BC94] das die Nichtbeachtung des `<con>`-Abschnitts Anfragen sehr ineffizient macht.

(Die – zusätzliche – Berücksichtigung von `<narr>` führt sogar zu einer Reduzierung der Precision um 30%.)

Die Betrachtung von Mehrwortgruppen kann diesen Effekt aber deutlich abmildern.

Zur Ermittlung von Mehrwortgruppen gibt es nun im Prinzip zwei Ansätze:

- Die **Betrachtung des Wortabstandes**

  Wenn die Wörter A und B eine bekannte Mehrwortgruppe bilden, und die Worte A und B in einem Text mit maximal n Worten dazwischen auftreten, dann interpretiert man dies als Auftreten der Mehrwortgruppe.

  Durch die Berücksichtigung der Reihenfolge und von Satzgrenzen kann dieser Ansatz noch verfeinert werden.

  Beispiel für Probleme:

  college junior, *junior college*, *junior in college*

- Die **Verwendung eines Parsers**, der die grammatikalische Satzstruktur ermittelt

  Der Parser versucht dabei einen gegebenen Satz auf eine der ihm bekannten möglichen Satzformen abzubilden.

  Das Problem ist dabei, dass sich u.U. nicht alle Sätze an die Grammatik halten, und dass die möglichen Satzformen ausgesprochen komplex sind.

Möglicher Aufbau eines Systems

Der Ablauf in einem vollständig automatisierten System:

- Text → NLP-Komponente → Repräsentation → „retrieval engine“

Die NLP-Komponente selbst arbeitet dabei nach folgendem Ablauf:

- TAGGER → PARSER → Begriffe/Konzepte

Dieses Verfahren kann gleichermaßen auf Anfragen und Texte angewendet werden.

In dem Beispielsystem wird zunächst eine Folge von Programmen ausgeführt:

- ein Programm, das den einzelnen Sprachteilen erklärende *Tags* hinzufügt,
- ein lexikonbasiertes Stammformreduktionsverfahren,
- ein syntaktischer Parser.

Im Anschluss werden bestimmte Typen von Mehrwortgruppen aus dem aufgebauten Syntaxbaum extrahiert und zusätzlich zu den einzelnen Wörten als Begriffe betrachtet.

Schliesslich wird für einen Anfragetext noch eine Synonymerweiterung nachgeschaltet.

Ein Beispiel für eine Ableitung eines Parsers [Cho99]

np = noun phrase = Hauptwortphrase; v = Verb, vp = verb phrase; det = determiner; final-punc = abschließendes Satzzeichen; proper-noun = Eigenname

sich bildender Satz	**angewendete Regel**		
s-maj			
s final-punc	s-maj	→	s final-punc
np vp final-punc	s	→	np vp
proper-noun vp final-punc	np	→	proper-noun
proper-noun v np final-punc	vp	→	v np
Jane v np final-punc	proper-noun	→	*Jane*
Jane likes np final-punc	v	→	*likes*
Jane likes det noun final-punc	np	→	det noun
Jane likes the noun final-punc	det	→	*the*
Jane likes the doll final-punc	noun	→	*doll*
Jane likes the doll.	final-punc	→	.

Probleme bei der Konstruktion eines Parsers für natürliche Sprache

- Im Idealfall benötigt man ein Wörterbuch, dass für jedes Wort seine möglichen Verwendungen auflistet.
  (z.B. die Online Version des Oxford Advanced Learner's Dictionary (OALD))
- natürliche Sprache ist sehr oft mehrdeutig hinsichtlich des möglichen Syntaxbaumes!
  Folge: der Parser findet einen Ableitungsbaum, aber nicht unbedingt den gemeinten!
- Um eine Sprache vollständig abzudecken, bräuchte man eine Unmenge an Regeln und Ausnahmen.
  Folge: Ein Parser kann manche Sätze nicht „verstehen“, muss danach aber wieder „sauber ausetzen“ können.
  Der Parser sollte also fehlertolerant sein, weil
  - Sätze in den Texten z.B. durch Tippfehler falsch aufgebaut sein könnten
  - die Grammatik des Parsers unvollständig sein wird.

Beispiel für einen Parser: der Tagged Text Parser (TTP) [Str94]

Dieser Parser basiert auf über 400 Regeln.

Bei Versuchen mit ca. 130 Mio. Wörtern aus der TREC-Kolektion (Wall Street Journal) schaffte der Parser auf einer SparcStation10 ca. 80 Wörter pro Sekunde.

Um die Verarbeitung zu beschleunigen wird der Text vor der Verarbeitung durch den Parser „zunächst getagged“.

Einige Beispiele für die Regeln zur Vergabe der Tags [Kow97]:

CLASS	EXAMPLES
determiners	a, the
singular nouns	paper, notation, structure, language
plural nouns	operations, data, processes
preposition	in, by, of, for
adjective	high, concurrent
present tense verb	presents, associates
present participal	multiprogramming

Anschließend wird eine Stammformreduktion durchgeführt.

Der Parser versucht nun hieraus sogenannte „Head-Modifier“ Strukturen zu erzeugen.

Der „Head“ ist dabei das zentrale Element des Paares.

Der „Modifier“ ist ein Zusatz oder eine nähere Bestimmung des „Head“ .

Beispiel für die Arbeitsweise des TTP:

(*dt* = determiner; *np* = Eigenname; *vbd* = tensed plural verb, *vbn* = past participle.)

INPUT SENTENCE
The former Soviet president has been a local hero ever since a Russian tank invaded Wisconsin.

TAGGED SENTENCE
The/*dt* former/*jj* Soviet/*jj* president/*nn* has/*vbz* been/*vbn* a/*dt* local/*jj* hero/*nn* ever/*rb* since/*in* a/*dt* Russian/*jj* tank/*nn* invaded/*vbd* Wisconsin/*np* ./*per*

TAGGED & STEMMED SENTENCE
the/*dt* former/*jj* soviet/*jj* president/*nn* have/*vbz* be/*vbn* a/*dt* local/*jj* hero/*nn* ever/*rb* since/*in* a/*dt* russian/*jj* tank/*nn* invade/*vbd* wisconsin/*np* ./*per*

PARSED SENTENCE

```
[assert
    [[perf [HAVE]] [[verb [BE]]
        [subject [np [n PRESIDENT] [t_pos THE] [adj [FORMER]] [adj [SOVIET]]]]
        [object [np [n HERO] [t_pos A] [adj [LOCAL]]]]
        [adv EVER]
        [sub_ord [SINCE
                [[verb [INVADE]]
                    [subject [np [n TANK] [t_pos A] [adj [RUSSIAN]]]]
                    [object [np [name [WISCONSIN]]]]]]]]]
```

EXTRACTED TERMS & WEIGHTS

president	2.623519	soviet	5.416102
president+soviet	11.556747	president+former	4.594883
hero	7.896426	hero+local	14.314775
invade	8.435012	tank	6.848128
tank+invade	17.402237	tank+russian	16.030809
russian	7.383342	wisconsin	7.785689

- BE ist hier das zentrale Prädikat, das durch HAVE näher bestimmt wird und 2 Argumente hat, ein Subjekt und ein Objekt, sowie 2 Attribute, ein Adverb und eine Subordination.
- INVADE ist ein Prädikat in der Subordination mit 2 Argumenten (Subjekt und Objekt).
- Das Subjekt zu BE ist ein zusammengesetztes Hauptwort mit PRESIDENT als Hauptbestandteil und zwei näheren Bestimmungen (FORMER und SOVIET).

Aus dieser Struktur können nun „Head-Modifier"-Paareabgeleitet werden, die Kandidaten für Mehrwortgruppen sind.

Im Allgemeinen werden folgende Typen von Paaren betrachtet:

1. ein Kopf-Substantiv einer Substantivphrase und sein linkes Adjektiv oder seine linke Substantivbeifügung,
2. ein Kopf-Substantiv und der Kopf seiner rechten Beifügung,
3. das zentrale Verb eines Halbsatzes und der Kopf seiner Objektphrase, und
4. der Kopf der Subjektphrase und das zentrale Verb.

Mit den obigen 4 Fällen erkennen wir den Begriff retrieve+information aus den folgenden Fragmenten:

information retrieval system, *retrieval of information from databases*, und *information that can be retrieved by a user-controlled interactive search process*.

Ein Problem hierbei ist die Mehrdeutigkeit der Sprache:

- Bei der Auswahl der Mehrwortgruppen wird auch die statistische Verteilung der verbundenen Begriffe betrachtet, um zu entscheiden, ob die Verbindung eines Hauptworts mit einem Adjektiv sowohl syntaktisch korrekt als auch semantisch relevant ist.
- So können wir z.B. *language+natural* und *processing+language* aus *natural language processing* ableiten.
- Aber wir sollten nicht *case+trading* aus *insider trading case* ableiten.
- Auf der anderen Seite sollten wir wieder *trading+insider* aus *insider trading sanctions act* oder *insider trading activity* ableiten.

Die Bestimmung der Wichtigkeit von Mehrwortgruppen

Die klassische *tf.idf* Formel hat laut Strzalkowski einige Probleme
(tf = term frequency; idf = inverse document frequency):

$$w_{dk} = \frac{tf_{dk} \cdot \log \frac{N}{n_k}}{\sqrt{\sum_{i=1}^{t} \left(tf_{di} \cdot \log \frac{N}{n_i} \right)^2}}$$

- Die Formel gewichtet relativ allgemeine Terme zu stark, denn nur solche werden sehr oft innerhalb eines Dokumentes vorkommen.
- Begriffe, die aufgrund ihrer Spezifität recht selten innerhalb eines Dokumentes vorkommen, werden nicht stark genug berücksichtigt.
- Das Problem der Abhängigkeiten zwischen Begriffen, das bei der Betrachtung von Mehrwortgruppen verstärkt auftritt, wird nicht berücksichtigt.
  So sind z.B. *launch+satellite* und *satellite* nicht unabhängig voneinander.

Zunächst werden für jedes Paar aus Anfrage und Dokument die
N Terme mit den höchsten idf-Werten, die in der Anfrage und im Dokument vorkommen, stärker gewichtet. Ferner werden die tf-Werte logarithmiert und mit einer Konstanten multipliziert.

Es ergibt sich folgende Formel:

$$weight(T_i) = (C_1 \cdot \log(tf) + C_2 \cdot \alpha(N, i)) \cdot idf$$

$\alpha(N, i)$ ist dabei

- 1, wenn der Term zu den N in Anfrage und Dokument vorkommenden Termen mit den höchsten idf-Werten gehört und
- 0 sonst.

Vergleichsmessungen

Erklärung:

Die Tabellenwerte beziehen sich auf 50 ad-hoc-Anfragen gegen die WSJ-Kollektion, wobei jeweils die 1000 „besten“ Dokumente ermittelt wurden.

- txt1 — einfache Begriffe aus den Feldern `<narr>` und `<desc>`
- txt2 — `<narr>` und `<desc>` Felder, wobei Terme mit geringen Gewichten nicht betrachtet wurden
- txt2+nlp — `<narr>` und `<desc>` Felder einschließlich Mehrwortgruppen mit der neuen Gewichtungsformel
- con — `<desc>` und `<con>` Felder; Terme mit geringen Gewichten nicht betrachtet; keine Mehrwortgruppen betrachtet
- con+nlp — `<desc>` und `<con>` Felder und Betrachtung von Mehrwortgruppen plus neue Formel

Run	txt1	txt2	txt2+nlp	con	con+nlp
Tot number of docs over all queries					
Rel	3929	3929	3929	3929	3929
RelRet	2736	3025	3108	3332	3401
%chg		+9.0	+14.7	+21.8	+24.3
Recall	(interp) Precision Averages				
0.00	0.6874	0.7318	0.7201	0.7469	0.8063
0.10	0.4677	0.5293	0.5239	0.5726	0.6198
0.20	0.3785	0.4532	0.4751	0.4970	0.5566
0.30	0.3060	0.3707	0.4122	0.4193	0.4786
0.40	0.2675	0.3276	0.3541	0.3747	0.4257
0.50	0.2211	0.2815	0.3126	0.3271	0.3828
0.60	0.1765	0.2406	0.2752	0.2783	0.3380
0.70	0.1313	0.1783	0.2142	0.2267	0.2817
0.80	0.0828	0.1337	0.1605	0.1670	0.2164
0.90	0.0451	0.0818	0.1014	0.0959	0.1471
1.00	0.0094	0.0159	0.0194	0.0168	0.0474

Run	txt1	txt2	txt2+nlp	con	con+nlp
Tot number of docs over all queries					
Rel	3929	3929	3929	3929	3929
RelRet	2736	3025	3108	3332	3401
%chg		+9.0	+14.7	+21.8	+24.3
Average precision over all rel docs					
Avg	0.2309	0.2835	0.3070	0.3210	0.3759
%chg		+22.8	+33.0	+39.0	+62.8
Precision at N documents					
5	0.5000	0.5240	0.5200	0.5600	0.6040
10	0.4080	0.4600	0.4900	0.5020	0.5580
100	0.2380	0.2790	0.2914	0.3084	0.3346
R-Precision (after Rel)					
Exact	0.2671	0.3053	0.3332	0.3455	0.3950
%chg		+14.3	+24.7	+29.3	+47.9

3.3.2 Mehrwortgruppen-Identifikation im Darmstädter Indexierungsansatz

Ein einfaches, robustes Verfahren zur Identifikation von Mehrwortgruppen ist im Rahmen der Arbeiten zum Darmstädter Indexierungsansatz [Lustig 86] entwickelt worden.

Dabei ist ein **Mehrwortgruppen-Wörterbuch** vorhanden, das **automatisch erstellt** wurde.

Bei der Analyse eines Textes wird beim Auftreten einer Komponente einer Mehrwortgruppe zunächst geprüft, ob die restlichen Komponenten ebenfalls **innerhalb eines vorgegebenen Maximalabstandes** im Text auftreten.

In dem folgenden Text werden unter anderem die untenstehenden Mehrwortgruppen identifiziert.

Die zu diesen Mehrwortgruppen gehörenden Komponenten sind im Text fettgedruckt:

Current-voltage spectra of metal/oxide/SnTe diodes. Pt. 1

In metal/oxide/SnTe **tunnel junctions** (where the oxide is Al_2O_3 or SiO_2 and the metal is lead or **aluminium**) on BaF_2 or NaCl **substrates** the **tunnel current** $I(U)$ and its derivatives $I'(U)$ and $I''(U)$ were **measured** at 4.2 K. Additionally the Hall **coefficient** and **electrical conductivity** of the monocrystalline SnTe **films** were determined at the same temperature. The pronounced oscillations in I'' suggest the existence of a quantum size effect in the very thin SnTe films in several cases, although this is complicated by various other processes. The most important features of the different types are discussed briefly.

- TUNNEL JUNCTION
- TUNNEL CURRENT
- ELECTRICAL CONDUCTIVITY
- ALUMINIUM SUBSTRATES
- MEASURE ELECTRICAL
- FILM COEFFICIENT

Wenn eine Mehrwortgruppe auf diese Art identifiziert wurde, dann werden die Werte folgender formaler Kriterien zur Beschreibung der Form des Vorkommens der Mehrwortgruppe verwendet:

1. Abstand zwischen den Komponenten (wobei bei der Bestimmung des Wortabstandes alle Wörter, nur Stoppwörter oder nur relevante Wörter gezählt werden können)
2. Reihenfolge der Komponenten (stimmt die Reihenfolge im Text mit der im Wörterbuch überein?)
3. Sind alle Komponenten im gleichen Satz?
4. Für jede Komponente: Besteht Grundformgleichheit mit dem Wörterbucheintrag, oder nur Gleichheit in der Stammform?

Entsprechend den Werten dieser Kriterien kann man nun Klassen von Vorkommensformen bilden.

Durch den Vergleich mit einer (intellektuell erstellten) korrekten Identifikation von Mehrwortgruppen kann man dann für jede Klasse die Wahrscheinlichkeit bestimmen, dass eine so identifizierte Mehrwortgruppe auch syntaktisch korrekt ist.

(im obigen Beispiel sind die ersten drei Mehrwortgruppen syntaktisch korrekt, die übrigen drei aber syntaktisch falsch.

Allerdings ist MEASURE ELECTRICAL zwar syntaktisch falsch, aber semantisch korrekt, so dass bezogen auf das Retrieval kein eigentlicher Fehler vorliegt.)

Es ist offensichtlich, dass dieses einfache Verfahren
für keine der betrachteten Klassen vollkommen korrekt ist.

Gleiches gilt aber für alle Arten von computerlinguistischen Verfahren.

Daher muss man versuchen, die hieraus resultierende Unsicherheit beim Retrieval entsprechend zu berücksichtigen.

Bei dem hier betrachteten Verfahren kann man die oben bestimmten Klassen-Wahrscheinlichkeiten jeweils den so identifizierten Mehrwortgruppen zuordnen.

Diese Werte können dann als Gewichtungen der Mehrwortgruppe bezüglich des Textes betrachtet werden und entsprechend in das Retrievalverfahren eingehen.

Experimentelle Untersuchungen haben ergeben, dass von den o.g. Merkmalen nur das erste und das vierte die Identifikationssicherheit signifikant beeinflussen, während die anderen Kriterien keinen nennenswerten Einfluss haben.

Ein Nachsatz zur Begriffswelt:

Präkombination: In der Indexierungssprache angelegte Begriffsverknüpfungen (Komposita oder Mehrwortausdrücke)
[d.h. Komposita dürfen in den Dokumenten nicht in anderen Formen vorkommen!]
Beispiele für Deskriptoren: Halswirbelfraktur, Europäischer Binnenmarkt

Präkoordination: Verknüpfungen die im Zuge der Indexierung hergestellt werden.
Deskriptoren: Halswirbel, Europa, Binnenmarkt, Fraktur
Indexierung: Halswirbel / Fraktur; Europa / Binnenmarkt

Postkoordination: Verknüpfungen während der Suche mittels Boole'scher oder sonstiger Operatoren.

Beispiele: Europa UND Binnenmarkt;
(Binnenmarkt ODER Währungsunion) UND Europa

3.4 Terminologische Kontrolle

Um die Mehrdeutigkeit der Sprache in den Griff zu bekommen, versucht man eine „eindeutige“ Begriffsverwendung zu erzwingen.

Die Idee kommt aus der manuellen Indexierung:

Indexierung / Indexieren als zweistufiger Prozess

1. das Erkennen der (wiederauffindbar zu machenden) Essenz eines Textes
2. das Wiedergeben dieser Essenz in einer ausreichend gut voraussehbaren und ausreichend wiedergabetreuen Form

Dazu verwendet man eine *künstliche Sprache* in der die verwendeten Bezeichnungen eindeutig auf einen definierten Begriff bezogen und Begriffe nur durch eine Bezeichnung repräsentiert sind.

Indexierungssprache

- Künstliche Sprache, mit deren Hilfe der Teil 2 des Indexierungsprozesses ausgeführt wird.
- Diese Sprachen sind geregelt, d.h. sie unterliegen einer terminologischen Kontrolle.
- Elemente einer Indexierungssprache werden Deskriptoren genannt.

Terminologiekontrolle:

Alle Regeln und Aktivitäten, welche einen Deskriptor eindeutig definieren und Mehrdeutigkeiten und Unklarheiten ausschalten (Synonyme, Homonyme, Polyseme, usw.).

Synonymkontrolle

Bei der Synonymkontrolle werden Bezeichnungen zu Äquivalenzklassen zusammengefasst.

Man kann folgende Arten von Synonymie unterscheiden:

- Schreibweisenvarianten
  Friseur – Frisör; UN – UNO – Vereinte Nationen
- unterschiedliche Konnotationen, Sprachstile, Verbreitung
  Telefon – Fernsprecher; Pferd – Gaul; Myopie – Kurzsichtigkeit
- Quasi-Synonyme
  Schauspiel – Theaterstück; Rundfunk – Hörfunk

Eine anderes Beispiel:

- Synonyme: Roß, Gaul, Mähre, Klepper, ...
- Deskriptor (Vorzugsbenennung): Pferd
- Begriff: Pferd

Im Thesaurus werden darüber hinaus Begriffe mit geringen oder irrelevanten Bedeutungsdifferenzen zu Äquivalenzklassen zusammengefasst:

- unterschiedliche Spezifität
  Sprachwissenschaft – Linguistik
- Antonyme
  Härte – Weichheit
- zu spezieller Unterbegriff
  Weizen – Winterweizen
- Gleichsetzung von Verb und Substantiv / Tätigkeit und Ergebnis
  Wohnen – Wohnung.

Die Entscheidung, ob zwei Begriffe als Quasisynonyme zu behandeln sind, hängt dabei immer von der jeweiligen Anwendung ab.

Polysemkontrolle

Bei der Polysemkontrolle werden mehrdeutige Bezeichnungen auf mehrere Äquivalenzklassen aufgeteilt.

Man kann hierbei noch unterscheiden zwischen

- *Homographen* (Beispiel: Tenor)
  - Tenor = Haltung; Inhalt, Sinn, Wortlaut
  - Tenor = hohe Männerstimme; Tenorsänger
- eigentlichen *Polysemen*
  - Bank = Sitzgelegenheit
  - Bank = Kreditanstalt

Zur terminologischen Kontrolle wird dabei i.a. ein Thesaurus verwendet:

„Ein Thesaurus ist eine natürlich-sprachlich basierte Dokumentationssprache, die die umkehrbar eindeutige Zuordnung von Begriffen und Bezeichnungen der natürlichen Sprache anstrebt, indem sie vollständige Vokabularkontrolle und terminologische Kontrolle ausübt und die Begriffe sowie Relationen zwischen ihnen durch Darstellung von Relationen zwischen den Bezeichnungen und gegebenenfalls zusätzliche Hilfsmittel darstellt.“ (DGD-KTS 1975)

Wichtig ist in diesem Zusammenhang der Begriff des *Deskriptors*:

- Calvin Mooers führte 1956 das Konzept des Deskriptors ein.
- Ein Deskriptor wurde nicht einfach als Wort aufgefaßt, sondern als Repräsentant eines Begriffs.
- Die Bedeutung wird durch Definition festgelegt und hat nur Gültigkeit innerhalb eines Systems.

Thesaurusrelationen (nach DIN 1643-1)

Es werden drei Grundtypen von Relationen unterschieden:

1. Äquivalenzrelationen
2. Hierarchierelationen
   (a) Abstraktionsrelation (generische Relation)
   (b) Bestandsrelation (partitive Relation)
3. Assoziationsrelationen

Darstellung von Relationen

wenn Abstraktions- und Bestandsrelationen nicht unterschieden werden:

Kurzzeichen		Bedeutung
D	E	
TT	TT	Top Term (Kopfbegriff einer Hierarchie)
OB	BT	übergeordneter Begriff
UB	NT	untergeordneter Begriff
VB	RT	verwandter Begriff (Assoziationsrelation)

wenn Abstraktions- und Bestandsrelationen unterschieden werden:

Kurzzeichen		Bedeutung
D	E	
OA	BTG	Oberbegriff (Abstraktionsrelation)
UA	NTG	Unterbegriff (Abstraktionsrelation)
SP	BTP	Verbandsbegriff (Bestandsrelation)
TP	NTP	Teilbegriff (Bestandsrelation)

(B = broader, N = narrower, T = term, G = generalization, P = paritipation)

Terminologiekontrolle:

Begriffszerlegung: Morphologische Zerlegung

Sprachliche Zerlegung in Wortbildungselemente (Morpheme)

- Abfallbeseitigung = Abfall UND Beseitigung
- Krankenhausbibliothek = Krankenhaus UND Bibliothek

Probleme:

- Eisenbahn = Eisen UND Bahn (?)
- Handschuh = ?

Terminologiekontrolle:

Begriffszerlegung: Semantische Zerlegung

Zerlegung in begriffliche Einheiten, die in Kombination den Ausgangsbegriff reproduzieren

- Eisenbahn = Schienenverkehr UND Überlandverkehr
- Handschuh = Hand UND Bekleidung

Beispiel für einen Thesaurus: *Global Legal Information Network*

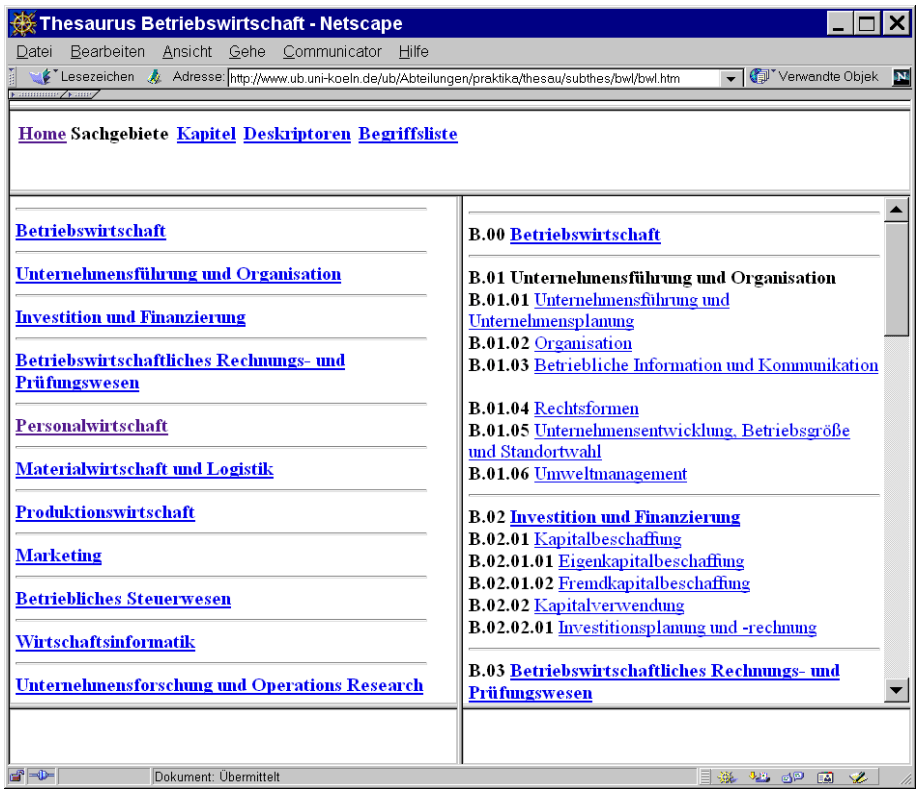
```
Children

Broader Term
    >Persons
    >>Civil code
Related Term
    Abortion
    Adoption
    Celiac Disease
    Civil code
    Educational law
    Guardian & ward
    Juvenile courts
    Juvenile delinquency
    Kidnapping
    Midwives
    Parent & child
    Parentage
    Support
    Women
    Youth
Aprv Stat Date
    28-Jul-9
```

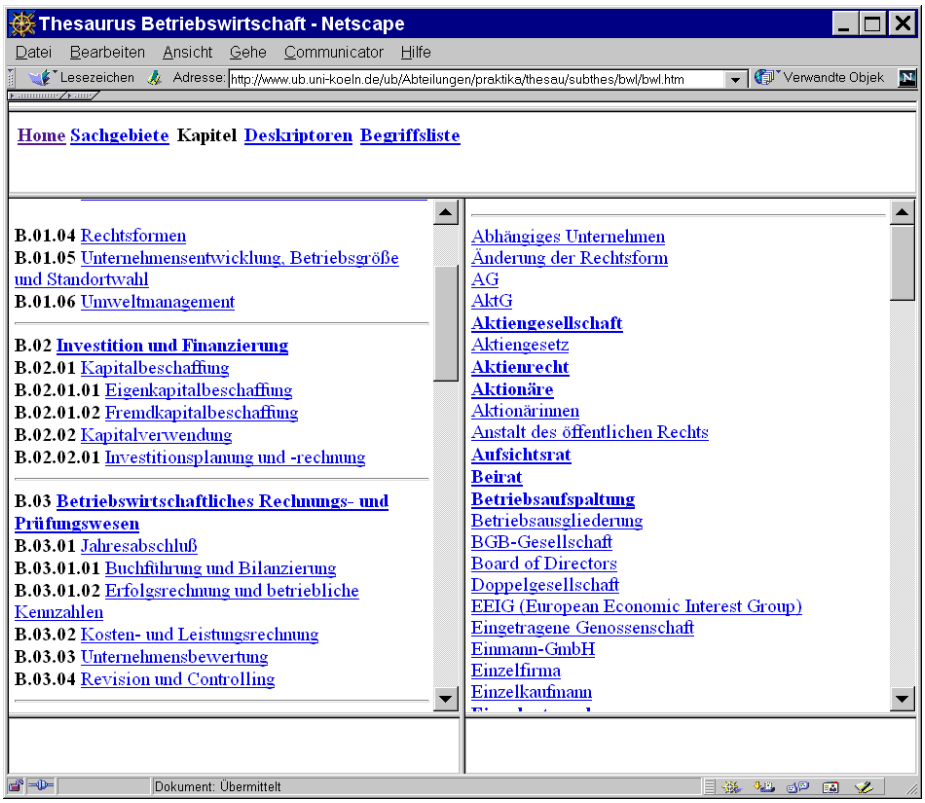
(`http://lcweb2.loc.gov/glin/indxhlp.html`)

Wirtschaftsthesaurus (`http://www.ub.uni-koeln.de/ub/Abteilungen/praktika/thesau/thesauru.htm`)

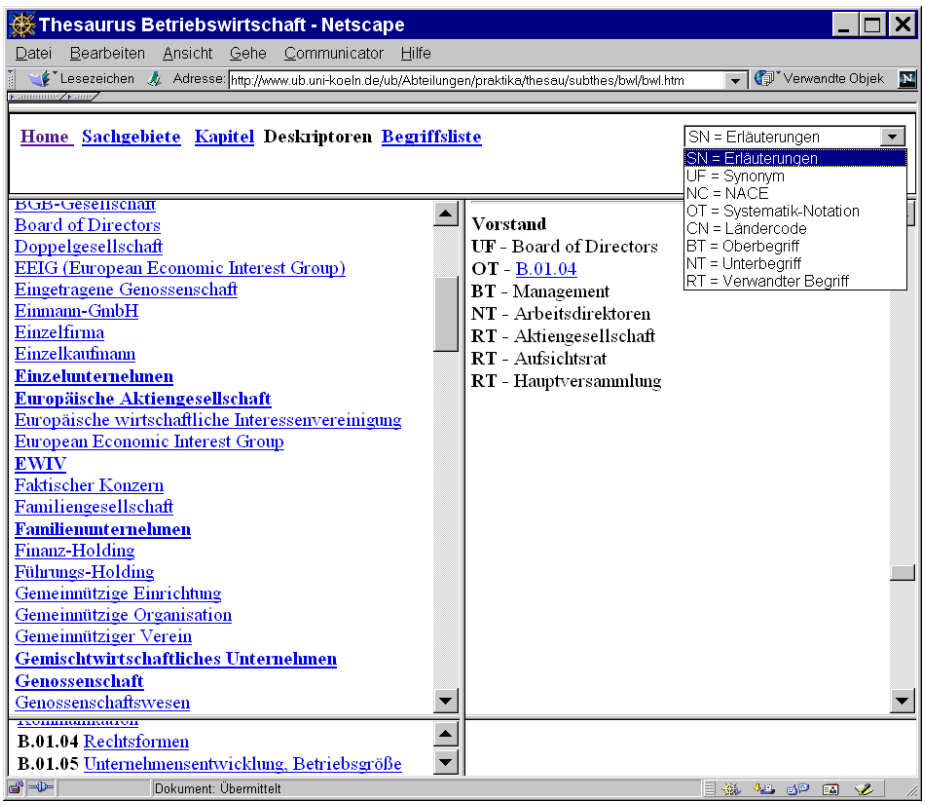
Auswahl der Betriebswirtschaft:



Auswahl von *B.01.04 Rechtsformen* im rechten Teilfenster:



Auswahl von *Board of Directors* im rechten Teilfenster:



Datenformat für Thesauruseinträge

- Deskriptor
- Übersetzter Begriff
- Nicht-Deskriptor
- Formel
- Oberbegriff
- Unterbegriff
- Benutze in Kombination
- Benutze Kombination
- Verwandter Begriff
- Definition (Scope Note)
- Verwendungshinweis
- Quelle
- Notation
- Einführungsdatum
- Häufigkeit der Indexierung
- Häufigkeit der Recheche
- Kommentar

Kapitel 4

Klassifikationen

Warum behandeln wir Klassifikationen?

- Klassifikationen verdeutlichen einige Grundprobleme des Information Retrieval.
- Klassifikationsschemata sind in vielen Bereichen notwendig (z.B. wenn Bücher physisch angeordnet werden müssen).
- Klassifikationsschemata sind häufig eine sinnvolle Ergänzung zu anderen IR-Systemen.

Ziel von Klassifikationen

Ordnung durch Klassenbildung i. Allg. unter Verwendung einer systematischen Notation.

Beispiele aus anderen Bereichen (nach [Mei97]):

- Stoffgruppen in der Chemie
- Taxonomien in der Biologie
- Stoffgliederungen in Enzyklopädien

In Bibliotheken ist eine systematische Klassifikation als Aufstellungssystematik zur Kennzeichnung der Bücher von Vorteil.

Beispiel: *Patentklassifikation (IPC)*

Recherche nach Internationaler Patentklassifikation (IPC)

Die Internationale Patentklassifikation (IPC) ist ein hierarchisches System mit folgendem Aufbau:

G	Sektion	Physik
G01	Klasse	Messen; Prüfen
G01R	Unterklasse	Messen elektrischer Größen; Messen magnetischer Größen
G01R 23	Gruppe	Anordnungen zum Messen von Frequenzen; Anordnungen zum Analysieren von Frequenzspektren
G01R 23/16	Untergruppe	Spektralanalyse; Fourier-Analyse

Die IPC unterliegt im Abstand von fünf Jahren einer Revision.

Bereits vorhandene Dokumente werden nicht umklassifiziert, so daß Veränderungen aufgrund einer zwischenzeitlichen Revision bei der Recherche berücksichtigt werden müssen.

(Nachschlagen in einer Konkordanz und in älteren Ausgaben der IPC.)

Die IPC wird erschlossen durch

- ein neunbändiges Handbuch (ein Band je Sektion und ein Einführungsband),
- ein Schlagwörterverzeichnis und
- eine CD-ROM-Datenbank.

Patentschriften erhalten

- eine Hauptnotation, die die Erfindung in ihrem Kern kennzeichnet, und
- ggf. weitere Nebennotationen, die weitere Aspekte bezeichnen.

Dementsprechend wird in Patentdatenbanken unterschieden nach Haupt-IPC (Main IPC, Feld ICM) und Neben-IPC (Secondary IPC, Feld ICS).

Die Eintragungen aus ICM und ICS sind - für die Recherche und für die Anzeige - gemeinsam im Feld IC verfügbar.

Probleme von Klassifikationen:

- insbesondere wissenschaftliche Dokumente behandeln oft innovative Gebiete

⇒ Gebiete sind u.U. in der Klassifikation nicht berücksichtigt

- wissenschaftliche Arbeiten behandeln gerne „interdisziplinäre“ Themen
  Beispiel: Titel „Datenbanken und Information Retrieval – zwei getrennte Welten?“

⇒ Eine eindeutige Zuordnung ist nicht möglich

- Zusammenfassend:
  - Klassifikationen sind von Natur aus *ein-dimensional*
  - Klassifikationen müssen fortentwickelt werden

Ein weiteres Beispiel: die Dezimalklassifikation (DK)

Um die Systematik zu vereinheitlichen, hat der amerikanische Bibliothekar **Melvil Dewey** (1851-1931) 1876 die Dezimalklassifikation geschaffen.

Die DK kennzeichnet jede Grundwissenschaft mit einer Zahl von 0-9 (Bsp.: 0:Allgemeines, 1:Philosophie, Psychologie, 2:Religion, Theologie usw.)

Diese Zahl kann durch Hinzufügen der Zahlen 0, 1, 2, 3, ..., 9 in 10 Untergruppen gegliedert werden.

Die Untergruppen können ihrerseits auf gleiche Weise wieder unterteilt werden.

Z.B. beim Gebiet 6:

- 62 Technik;
- 622 Bergbautechnik;
- 622.3 Einzelne Bergbauzweige;
- 622.33 Kohlebergbau.

Main classes

000	Generalities
100	Philosophy and related disciplines
200	Religion
300	The social sciences
400	Language
500	Pure sciences
600	Technology (applied sciences)
700	The arts
800	Literature
900	General geography and history

Main divisions of the technology class

600	Technology (applied sciences)
610	Medical sciences
620	Engineering and allied operations
630	Agriculture and related technologies
640	Home economics and family living
650	Management and auxiliary services
660	Chemical and related technologies
670	Manufactures
680	Manufacture for specific uses
690	Buildings

Main divisions of the engineering class

620	Engineering and allied operations
621	Applied physics
622	Mining and related operations
623	Military and nautical engineering
624	Civil engineering
625	Railroads, roads, highways
626	[not used]
627	Hydraulic engineering
628	Sanitary and municipal engineering
629	Other branches of engineering

Die Haupttafel der sytematischen Einteilung wird durch Hilfstafeln mit Anhängezahlen ergänzt.

Die Anhängezahlen dienen der Facetierung, d.h.

- der Untergliederung nach Ort, Zeit, Form, Sprache u.a.
- oder auch noch zur feineren Stoffgliederung.

Z.B. bedeutet (430) Deutschland ⇒ 622(430) Bergbautechnik in Deutschland.

Beziehungen zweier Begriffe werden durch Doppelpunkt ausgedrückt, z.B. 621.3:622 Elektrotechnik im Bergbau.

Beispiele:

- 53(038) — Dictionary of physics
  - Die 53 steht hier für Physik.
  - (038) steht für Wörterbuch (Dictionary).
  - Man beachte, dass die Notation (0...) die Form festlegt.
- 622 + 629 — Mining and metallurgy
  - Das + wird verwendet, um die Verbindung zwischen zwei Themengebieten zu kennzeichnen.
- [23/28:294.3](540) — Christianity in relation to Buddism in India
  - Dabei bezeichnet 23/28 die christlichen Religionen.
    Der „/“ wird dabei verwendet, um ein Intervall von Themenbereichen anzugeben.
  - „:“ bezeichnet die symmetrische Beziehung zwischen zwei Sachverhalten.
  - 294.3 steht für den Buddhismus.
  - „[...]“ steht für eine Gruppierung von Themen.
  - (540) steht für Indien. (Man beachte, dass Ortsangaben immer in Klammern stehen.)

Zusammenfassung facettierende Elemente

Zur Facettierung in der DK dienen die Anhängezahlen, die durch spezielle Zeichen eingeleitet werden.

Es gibt einerseits allgemeine Anhängezahlen, die überall in der DK verwendet werden dürfen, und andererseits spezielle Anhängezahlen, die nur für bestimmte Klassen innerhalb der DK erlaubt sind.

Beispiele für allgemeine Anhängezahlen sind folgende (die jeweils einleitende Zeichenfolge ist vorangestellt):

=	Sprache
(0...)	Form
(...)	Ort
(=...)	Rassen und Völker
„...“	Zeit
.00	Gesichtspunkt
-05	Person

Verknüpfung von DK-Zahlen

Zur Verknüpfung von DK-Zahlen gibt es als syntaktische Elemente spezielle Sonderzeichen:

+	Aufzählung mehrerer Sachverhalte,
:	symmetrische Beziehung zwischen zwei Sachverhalten
::	asymmetrische Beziehung zwischen zwei Sachverhalten,
/	Erstreckungszeichen (zur Zusammenfassung mehrerer nebeneinanderstehender DK-Zahlen),
’	Zusamenfassungszeichen zur Bildung neuer Sachverhalte aus der Kombination einzelner DK-Komponenten.

Einordnung hierarchische Klassifikationen

Hierarchische Systeme zeichnen sich dadurch aus, daß eine stufenweise Spezifizierung der Klassen stattfindet.

Grundsätzlich muß jedes Element einer untergeordneten Klasse alle Merkmale der Elemente der übergeordneten Klasse mit zusätzlichen Merkmalen enthalten.

Diese enumerativen, also aufzählenden Klassifikationen wurden und werden auch heute noch als ordnungsschaffende Methode verwendet.

Sie veralten im Augenblick des Entstehens.

Die rasche Entwicklung der Wissenschaften macht ständige Änderungen und Anpassungen nötig, wodurch Inhomogenität und Brüche unvermeidlich werden.

Hinzu kommt die nicht zu verhindernde Inkonsistenz in der Notationsvergabe und die große Fehlerquelle beim Umgang mit Zahlenketten.

Klassifikationen müssen in unterschiedlichen Zusammenhängen und verschiedener Bedeutung gesehen werden.

- Einmal zur Wissensdarstellung und Wissensvermittlung, also wissens- und erkenntnisorientiert,
- zum anderen als Organisationsmittel zum geordneten Ablegen und Aufbewahren,
- sowie als inhaltsbeschreibendes Element, das bedeutet pragmatisch-dokumentarisch orientiert.

Klassifikationssysteme sind also mehr oder weniger komplexe Systeme von Themenklassen mit Ordnungscharakter.

Eine Themenklasse ist eine Zusammenfassung von gleichen Sachverhalten unter bestimmten Voraussetzungen.

Unter einem **monohierarchischen Klassifikationssystem** versteht man, dass eine Klasse nur einer anderen untergeordnet werden kann, während in **polyhierarchischen** Systemen eine Klasse mehreren anderen Klassen subsumiert wird.

Anfragesysteme

Varianten:

- Pattern Matching
- Adressierung der Semantik des Dokumentes
  - Boolesches Retrieval
  - Vektorraummodell
  - Probabilistisches Information Retrieval
  - ...

Kapitel 5

Pattern Matching in Texten

- Suche nach Zeichenketten in Texten
- Einfachste Möglichkeit des Information Retrieval
- Basistechnologie für viele anspruchsvollere Verfahren

Informelle Darstellung von Arten von String-Matching-Problemen:

Exaktes String-Matching: Wo tritt ein String *pat* in einem Text *text* auf? Beispiel: `fgrep` (fast grep, keine reg. Ausdrücke aber schnell)

Matching von Wortmengen: Gegeben sei eine Menge S von Strings. Wo tritt in einem Text ein String aus S auf? Beispiel: `agrep` (approximate grep)

Matching regulärer Ausdrücke: Welche Stellen in einem Text passen auf einen regulären Ausdruck? Beispiel: `grep`, `egrep` (expression grep)

Approximatives String-Matching: Welche Stellen in einem Text passen am besten auf ein Muster? (Best-Match-Anfrage)

Welche Stellen in einem Text stimmen mit einem Muster bis auf d Fehler überein? (Distance-Match-Anfrage) Beispiel: `agrep`

Editierdistanz: Wie kann man am „günstigsten“ einen String s in einen String t überführen? Beispiel: `diff`

Wozu braucht man String-Matching in IR-Systemen?

- Zur Überprüfung von Kandidaten.
  Manche Indexstrukturen (z.B. Signature Files, N-Grams) sind inexakte Filter, d.h. sie liefern u.U. zu einem Anfragestring Dokumente, die diesen String nicht enthalten.

- Zur Unterstützung von leistungsfähigen Operatoren (z.B. Linkstrunkierung oder Phrasensuche).
  Indexstrukturen liefern hier nur eine Kandidatenmenge.
  Insbesondere bei Distance- oder Best-Match-Anfragen ist der Einsatz von entsprechenden String-Matching-Algorithmen notwendig (vgl. *Glimpse*).

- Für kleine Datenbanken ohne bzw. nur mit sequentiellem Index (vgl. `grep` und *Glimpse*).
  Leistungsfähige Indexstrukturen benötigen sehr viel Speicherplatz (das drei- bis zehnfache der Rohdaten). Dies lohnt sich für kleine Datenbanken oft nicht.

Bezeichnungen:

- Ein Alphabet Σ ist eine endliche Menge von Symbolen. $|\Sigma|$ bezeichnet die Kardinalität von Σ.
- Ein String (*Zeichenkette*, *Wort*) s über einem Alphabet Σ ist eine endliche Folge von Symbolen aus Σ. $|s|$ bezeichnet die Länge von s.
- ϵ bezeichnet den leeren String.
- Wenn x und y Strings sind, dann bezeichnet xy die Konkatenation von x und y.
- $s[i]$ bezeichnet das i-te Element eines Strings s $(1 \leq i \leq |s|)$.
- $s[i \ldots j]$ bezeichnet den String $s[i]s[i+1] \ldots s[j]$. Für $i > j$ gelte $s[i \ldots j] = \epsilon$.
- Für einen String s (mit $m = |s|$) bezeichnet s^R die Umkehrung $s[m]s[m-1] \ldots s[1]$ von s.
- Für zwei Strings x und y gilt $x = y$ genau dann, wenn $|x| = |y| = m \wedge \forall i \in \{1, \ldots, m\} : x[i] = y[i]$.
- Wenn $\omega = xyz$ ein String ist, dann ist x ein Präfix und z ein Suffix von ω.
  Gilt $\omega \neq x$ $(\omega \neq z)$, dann ist x (z) ein echter Präfix (echter Suffix) von ω.

- Ein String x (mit $m = |x|$) heisst Substring (Faktor) von y, wenn ein i existiert mit $x = y[i \ldots i + m - 1]$.
  Andere Sprechweisen: x tritt in y an Position i auf, bzw. Position i ist ein Match für x in y.
- x (mit $m = |x|$) heisst Subsequenz von y, wenn Positionen $i_1 < i_2 \cdots < i_m$ existieren mit $x = y[i_1]y[i_2] \ldots y[i_m]$.

5.1 Exaktes String-Matching

Problem *Exaktes String-Matching*:

Gegeben sind die Strings *pat* und *text*.

(a) Man bestimme, ob *pat* ein Substring von *text* ist.

(b) Man bestimme die Menge aller Positionen, an denen *pat* in *text* auftritt
Diese Menge wird mit *MATCH(pat, text)* bezeichnet.

Bemerkungen:

- Im folgenden wird nur die Variante (a) des Problems betrachtet.
- Algorithmen für die Variante (b) erhält man durch einfache Modifikationen der Algorithmen für (a).

Bezeichnung: Im folgenden sei $m = |pat|$ und $n = |text|$.

5.1.1 Der naive Algorithmus

Der naive Ansatz besteht darin, für jede Position von *text* (bzw. solange *pat* ab der aktuellen Position in *text* passt) von neuem zu testen, ob *pat* an dieser Position auftritt.

Das allgemeine Schema für solch einen naiven Algorithmus lautet:

for $i := 1$ **to** $n - m + 1$ **do**
    man prüfe, ob $pat = text[i \ldots (i + m - 1)]$ gilt

Die Prüfung kann nun „von links nach rechts“ oder „von rechts nach links“ erfolgen.

Dies führt zu unterschiedlichen naiven Algorithmen und darauf aufbauend zu unterschiedlichen Ansätzen der Verbesserung.

Zunächst wird die Variante „von links nach rechts“ betrachtet.

Algorithmus zum naiven String-Matching von links nach rechts:

$i := 1$;
while $i \leq n - m + 1$ **do**
    /* Vergleiche ab Position i im *text* mit *pat* */
    while $j \leq m$ **and** $pat[j] = text[i + j - 1]$ **do** $j := j + 1$ **end**
    if $j = m + 1$ **then return true**;
    $i := i + 1$;
end
return false;

Aufwand:

Der naive Algorithmus löst das gegebene Problem in Zeit $O(nm)$ und Platz $O(m)$.

Genauere Abschätzung des zu erwartenden Aufwands:

Zu erwartende Zahl der Vergleiche an einer Position i:

- 1. Vergleich $\rightarrow$ immer $\Rightarrow P = 1$
- 2. Vergleich $\rightarrow$ nur, wenn 1. Zeichen gleich war $\Rightarrow P = \frac{1}{|\Sigma|}$
- 3. Vergleich $\rightarrow$ nur, wenn 1. + 2. Zeichen gleich war $\Rightarrow P = \frac{1}{|\Sigma|^2}$

  $\vdots$

$\Rightarrow$ für die zu erwartende Zahl von Vergleichen an einer Position i ergibt sich:

$$\begin{aligned} 1 + \tfrac{1}{|\Sigma|} + \tfrac{1}{|\Sigma|^2} + \cdots + \tfrac{1}{|\Sigma|^{m-1}} &= \Sigma_{j=0}^{m-1} \tfrac{1}{|\Sigma|^j} \\ &= \frac{\frac{1}{|\Sigma|^m}}{\frac{1}{|\Sigma|}-1} - \frac{1}{\frac{1}{|\Sigma|}-1} \\ &= \left(\tfrac{1}{|\Sigma|^m} - 1\right) \cdot \left(\tfrac{|\Sigma|}{1-|\Sigma|}\right) \\ &= \left(1 - \tfrac{1}{|\Sigma|^m}\right) \cdot \tfrac{|\Sigma|}{|\Sigma|-1} \end{aligned}$$

i nimmt in der Schleife $n - m + 1$ Werte an.

Damit ergibt sich für die Gesamtzahl der Vergleiche ein Erwartungswert von:

$$\frac{|\Sigma|}{|\Sigma| - 1} \cdot \left(1 - \frac{1}{|\Sigma|^m}\right)(n - m + 1) + O(1)$$

Beispiel: $|\Sigma| = 20$ und $m = 5$:

$$\begin{aligned} \tfrac{|\Sigma|}{|\Sigma|-1} \cdot \left(1 - \tfrac{1}{|\Sigma|^m}\right) &= \tfrac{20}{20-1} \cdot \left(1 - \tfrac{1}{20^5}\right) \\ &= 1,052631579\ldots \cdot \left(1 - \tfrac{1}{3200000}\right) \\ &= 1,052631250\ldots \end{aligned}$$

Bemerkungen:

- Für $pat = a^{m-1}b$ und $text = a^n$ benötigt der Algorithmus $(n-m+1)m = nm - m^2 + m$ Zeichenvergleiche (Worst Case).
- Bei einem zweielementigen Alphabet ($|\Sigma| = 2$) und zufällig erzeugten *pat* und *text* (jedes Zeichen unabhängig und jedes Symbol mit Wahrscheinlichkeit 1/2) ergibt sich für die durchschnittliche Anzahl an Zeichenvergleichen:

$$\frac{2}{2-1} \cdot \left(1 - \frac{1}{2^m}\right)(n-m+1) + O(1) \;=\; 2 \cdot \left(1 - \frac{1}{2^m}\right)(n-m+1) + O(1)$$

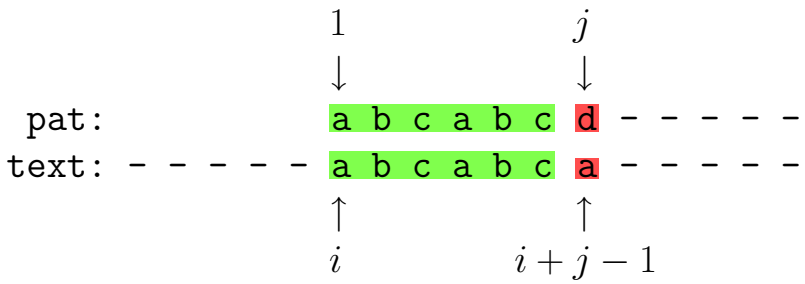
$$\approx \left(2 - 2^{-m+1}\right) n + O(1)$$

- Trotz der im Durchschnitt linearen Laufzeit lohnt sich der Einsatz von „besseren“ String-Matching-Algorithmen, denn:
  - die nachfolgenden String-Matching-Algorithmen haben sich nicht nur in der Theorie, sondern auch in der Praxis als erheblich effizienter erwiesen, und
  - ein gleichverteiltes Auftreten von Zeichen ist in der Realität extrem selten

5.1.2 String-Matching nach Knuth, Morris und Pratt

Der bisherige Algorithmus ist naiv im folgenden Sinn: nach jedem Mismatch an Stelle j von *pat* wird vergessen, dass *pat* bis zur Position $j-1$ auf *text* passt.

Kommt es an Stelle j von *pat* zu einem Mismatch, so gilt:
$pat[1 \ldots (j-1)] = text[i \ldots (i+j-2)]$



Dies kann wie folgt ausgenutzt werden:

Angenommen, *pat* tritt in *text* an einer Position $i+s$ mit $i < i+s < i+j-1$ auf.
Dann muss gelten: $pat[1 \ldots (j-s-1)] = pat[(1+s) \ldots (j-1)]$

Veranschaulichung:

```
 pat:           a b c a b c d - - - - - -  ← gescheiterter Vergleichsversuch
       pat:           a b c a - - - - - -  ← möglicher neuer Aufsetzpunkt
text: - - - - - a b c a b c a - - - - - -
                ↑     ↑     ↑
                i   i+s   i+j-1
```

s ist hier der Betrag, um den pat nach rechts verschoben wird.

Nach einem Mismatch an Position j von pat kann nur dann ab Position $i+s$ im Text ein Match beginnen, wenn $pat[1 \ldots (j-s-1)]$ ein Suffix von $pat[1 \ldots (j-1)]$ ist.

Im obigen Beispiel, sind die Bereiche, die übereinstimmen müssen, markiert.

Veranschaulichung:

```
       pat:           a b c a - - - - - -
 pat:           a b c a b c d - - - - - -
text: - - - - - a b c a b c a - - - - - -
                ↑     ↑     ↑
                i   i+s   i+j-1
```

Wenn wir also beim Vergleich mit der Stelle $i+j-1$ im Text eine Ungleichheit festgestellt haben, können wir

- an dieser Stelle $(i+j-1)$ des Textes mit dem Vergleichen fortfahren
- wir dürfen aber mit dem Pattern nicht an der Position 1 beginnen, sondern
- müssen die Position k (mit $k < j$) im Pattern nehmen, zu der $pat[1 \ldots (k-1)]$ ein Suffix von $pat[1 \ldots (j-1)]$ ist.

k ist dabei im Sinne der Veranschaulichung $j - s$ ($s = 3, j = 7, k = 4$).

Damit man keinen Match verpasst, muss s möglichst klein und somit k möglichst groß gewählt werden.

Konsequenzen für einen verbesserten Algorithmus:

- Man ermittle in einer Preprocessingphase zu jedem $j \in \{1, \ldots, m\}$ das größte k, so dass $pat[1 \ldots (k-1)]$ echter Suffix von $pat[1 \ldots (j-1)]$ ist.
  Der entsprechende Betrag wird mit $border[j]$ bezeichnet.

$$border[j] := \max_{1 \leq k \leq j-1} \{k | pat[1 \ldots (k-1)] = pat[(j-k+1) \ldots (j-1)]\}$$

  bzw.

$$border[j] := \max_{1 \leq k \leq j-1} \{k | pat[1 \ldots (k-1)] \text{ ist echter Suffix von } pat[1 \ldots (j-1)]\}$$

  Weiterhin gelte $border[1] = 0$
- Im Algorithmus schiebe man bei einem Mismatch an Position j des Pattern dieses um $s = j - border[j]$ Stellen nach rechts.
  D.h. man vergleicht die aktuelle Stelle in $text$ nun mit der Stelle $border[j]$ in pat.
- Durch die Maximalität ist s ein „safe shift“.

Algorithmus von Morris und Pratt

$i := 1$; /* gibt die Vergleichsposition im Text an */
$j := 1$; /* gibt die Vergleichsposition im Pattern an */
while $i \leq n - m + 1$ **do**
    while $j \leq m$ **and** $pat[j] = text[i + j - 1]$ **do**
        $j := j + 1$;
    end
    if $j = m + 1$ **then return true**;
    $i := i + j - border[j]$; /* neue Startpos. für potentielles Match im Text */ (A)
    $j := \max(border[j]; 1)$; /* $pat[1 \ldots (j-1)]$ ist ohnehin gleich $text[i \ldots (i+j-2)]$
        $\Rightarrow$ Vergleich erst ab $text[i + j - 1]$ */ (B)
end
return false;

Laufzeit des Algorithmus von Morris und Pratt

Wenn $border[j]$ (für $1 \leq j \leq m$) bereits vorliegt, löst der Algorithmus das Problem in Zeit $O(n)$.

Beweis:

- Es gibt höchstens $(n - m + 1) \in O(n)$ nicht erfolgreiche Vergleiche (nämlich höchstens einer für jedes i, d.h. für jede Startposition im Text).
- Man betrachte nun den Term $i + j$. Es gilt $2 \leq i + j \leq n + 1$.
- Immer, wenn der Vergleich $pat[j] = text[i + j - 1]$ erfolgreich war, wird $i + j$ um eins erhöht.
- Insgesamt wird $i + j$ in (A) und (B) nicht verringert.

$\Rightarrow$ Es gibt $\leq n$ erfolgreiche Vergleiche

$\Rightarrow$ und somit $O(n)$ Vergleiche insgesamt.

Es bleibt das Problem, $border[j]$ berechnen zu müssen.

Zunächst ein Beispiel für $border[j]$.

Definition Fibonacci-Strings:

Der n-te Fibonacci-String F_n ($n \geq 0$) ist wie folgt definiert:

- $F_0 = \epsilon$
- $F_1 = b$
- $F_2 = a$
- $F_n = F_{n-1}F_{n-2}$ für $n > 2$

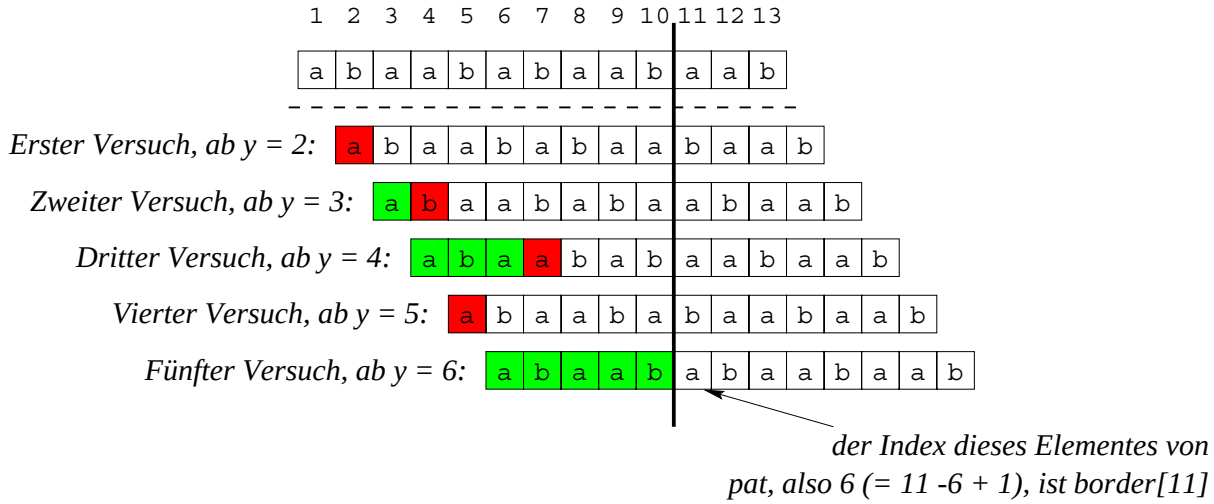
Beispiel:

$border[j]$ lautet für F_7:

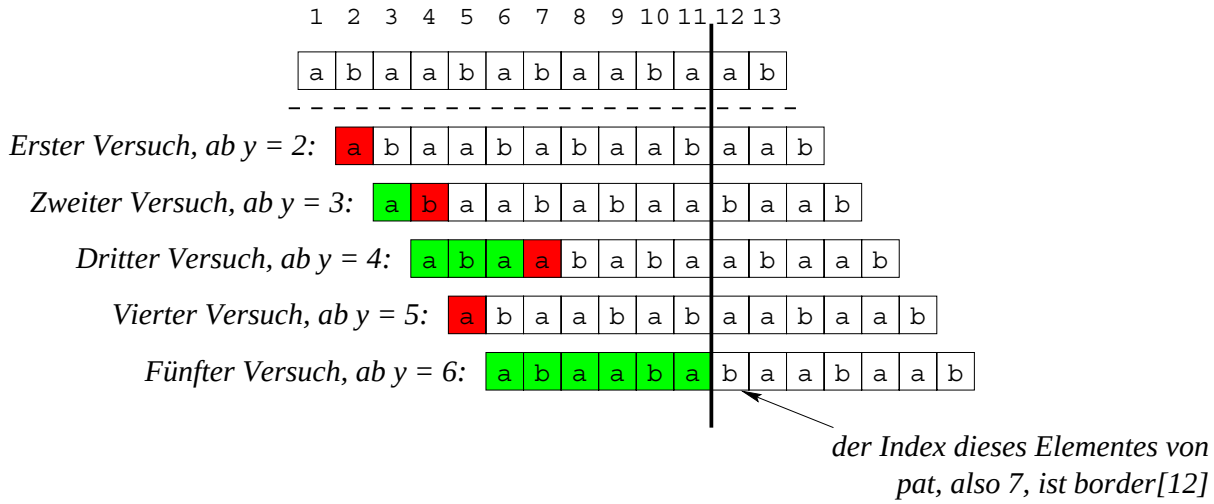
j	1	2	3	4	5	6	7	8	9	10	11	12	13
$pat[j]$	a	b	a	a	b	a	b	a	a	b	a	a	b
$border[j]$	0	1	1	2	2	3	4	3	4	5	6	7	5

Einfacher Ansatz zur Berechnung der Werte in *border*:

- Zur Berechnung von $border[x]$ prüfen wir für die Werte $y = 2, \ldots, x-1$ aufsteigend, ob an der Stelle y ein Suffix beginnt, dass gleichzeitig ein Präfix von *pat* ist.
- Formal: Wir prüfen, ob $pat[y \ldots (x-1)]$ ein Präfix von *pat* ist.
- $border[j]$ ist dann für das erste y, das dies erfüllt, $j - y + 1$.
- Zur Berechnung von $border[11]$ ergibt sich folgender Ablauf:



- Zur Berechnung von $border[12]$ ergibt sich folgender Ablauf:



- Man erkennt, dass sehr viele Überprüfungen mehrfach vorgenommen werden.
- Das vermeidet der folgende Ansatz, der nacheinander alle möglichen Startpositionen für ein echtes Suffix betrachtet.

  Für jede Startposition wird die Betrachtung fortgesetzt, bis zum ersten mal Ungleichheit besteht. Dann wird die nächste Startposition betrachtet.

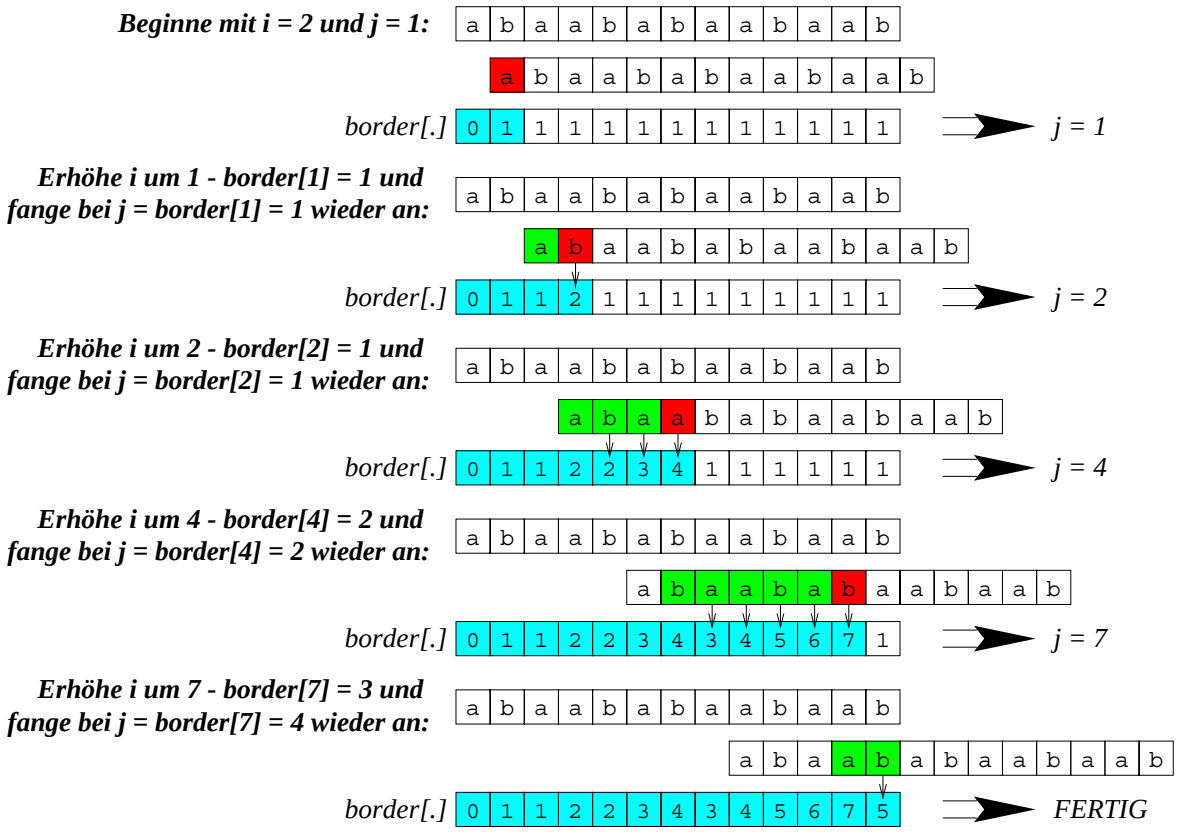
Die Berechnung der Tabelle `border` nutzt eine spezielle Version des Algorithmus von Morris und Pratt.

Man nutzt dabei die aus dem Beweis bekannte Tatsache, dass der Betrag $i + j$ nie verringert wird.

Algorithmus zur Berechnung der Tabelle `border`:

$border[1] := 0;$
for $j := 2$ **to** m **do** $border[j] := 1$ **end**;
$i := 2$; /* Startposition für den Suffix */
$j := 1$; /* Startposition für den Präfix */
while $i \leq m$ **do**
    while $i + j - 1 \leq m$ **and** $pat[j] = pat[i + j - 1]$ **do**
        /* Es gibt eine Übereinstimmung zwischen $pat[1 \ldots j]$ und $pat[i \ldots (i + j - 1)]$ */
        $j := j + 1;$
        $border[i + j - 1] := j$; /* Die Aufsetzposition j (jetzt um 1 erhöht) wird eingetragen */
    end
    $i := i + j - border[j]$; /* neue Startpos. für potentielles Match im Rest des Patterns */
    $j := \max(border[j]; 1)$; /* $pat[1 \ldots (j - 1)] = pat[i \ldots (i + j - 2)] \Rightarrow$ Vergleich ab $pat[i + j - 1]$ */
end

Der Algorithmus benötigt zur Berechnung der Tabelle `border` höchstens $O(m)$ Vergleiche.



		1	2	3	4	5	6	7	8	9	10	11	12	13
	pat	a	b	a	a	b	a	b	a	a	b	a	a	b
	border	0	1	1	1	1	1	1	1	1	1	1	1	1
	Zeiger	j	i											
$pat[j] = pat[i+j-1]$?	$pat[1] \neq pat[2+1-1] \Rightarrow$ Überspringen der While-Schleife													
$i := i + j - border[j]$;	Zeiger	j		i										
$j := \max(border[j]; 1)$;	Zeiger	j		i										
$pat[j] = pat[i+j-1]$?	$pat[1] = pat[3+1-1] \Rightarrow$ Eintritt in die While-Schleife													
$j := j + 1$;	Zeiger		j	i										
$border[i+j-1] := j$;	*border*	0	1	1	2	1	1	1	1	1	1	1	1	1
$pat[j] = pat[i+j-1]$?	$pat[2] \neq pat[3+2-1] \Rightarrow$ Überspringen der While-Schleife													
$i := i + j - border[j]$;	Zeiger		j		i									
$j := \max(border[j]; 1)$;	Zeiger	j			i									
$pat[j] = pat[i+j-1]$?	$pat[1] = pat[4+1-1] \Rightarrow$ Eintritt in die While-Schleife													
$j := j + 1$;	Zeiger		j		i									
$border[i+j-1] := j$;	*border*	0	1	1	2	2	1	1	1	1	1	1	1	1
$pat[j] = pat[i+j-1]$?	$pat[2] = pat[4+2-1] \Rightarrow$ Eintritt in die While-Schleife													
$j := j + 1$;	Zeiger			j	i									
$border[i+j-1] := j$;	*border*	0	1	1	2	2	3	1	1	1	1	1	1	1

		1	2	3	4	5	6	7	8	9	10	11	12	13
	pat	a	b	a	a	b	a	b	a	a	b	a	a	b
	border	0	1	1	2	2	3	1	1	1	1	1	1	1
	Zeiger			j	i									
$pat[j] = pat[i+j-1]$?	$pat[3] = pat[4+3-1] \Rightarrow$ Eintritt in die While-Schleife													
$j := j + 1$;	Zeiger				j, i									
$border[i+j-1] := j$;	*border*	0	1	1	2	2	3	4	1	1	1	1	1	1
$pat[j] = pat[i+j-1]$?	$pat[4] \neq pat[4+4-1] \Rightarrow$ Überspringen der While-Schleife													
$i := i + j - border[j]$;	Zeiger				j		i							
$j := \max(border[j]; 1)$;	Zeiger		j				i							
$pat[j] = pat[i+j-1]$?	$pat[2] = pat[6+2-1] \Rightarrow$ Eintritt in die While-Schleife													
$j := j + 1$;	Zeiger			j			i							
$border[i+j-1] := j$;	*border*	0	1	1	2	2	3	4	3	1	1	1	1	1
$pat[j] = pat[i+j-1]$?	$pat[3] = pat[6+3-1] \Rightarrow$ Eintritt in die While-Schleife													
$j := j + 1$;	Zeiger				j		i							
$border[i+j-1] := j$;	*border*	0	1	1	2	2	3	4	3	4	1	1	1	1

...

Der Algorithmus von Morris und Pratt kann noch effizienter gemacht werden.

Man betrachte folgendes Beispiel:

```
 pat:           a b a a b a - -
text: - - - - - a b a a b c - -
```

Es kommt für $j = 6$ zu einem Mismatch. Wegen $border[6] = 3$ wird pat um drei Zeichen nach rechts geschoben.

```
      pat:           a b a a b a - -
text: - - - - - a b a a b c - -
```

Es kommt also an der gleichen Stelle des Textes direkt wieder zu einem Mismatch!

Dies war abzusehen, denn für pat gilt: $pat[6] = pat[border[6]]$

Offensichtlich kann die bisher genutzte Bedingung, dass $pat[1 \ldots (k-1)]$ Suffix von $pat[1 \ldots (j-1)]$ ist, um die Bedingung $pat[k] \neq pat[j]$ verschärft werden.

Konsequenzen für eine weitere Verbesserung:

- man benutze statt $border[j]$ die folgende Variante $sborder[j]$:

$$sborder[j] := \max_{1 \leq k \leq j-1} \{k | pat[1 \ldots (k-1)] = pat[(j-k+1) \ldots (j-1)] \wedge pat[k] \neq pat[j]\}$$

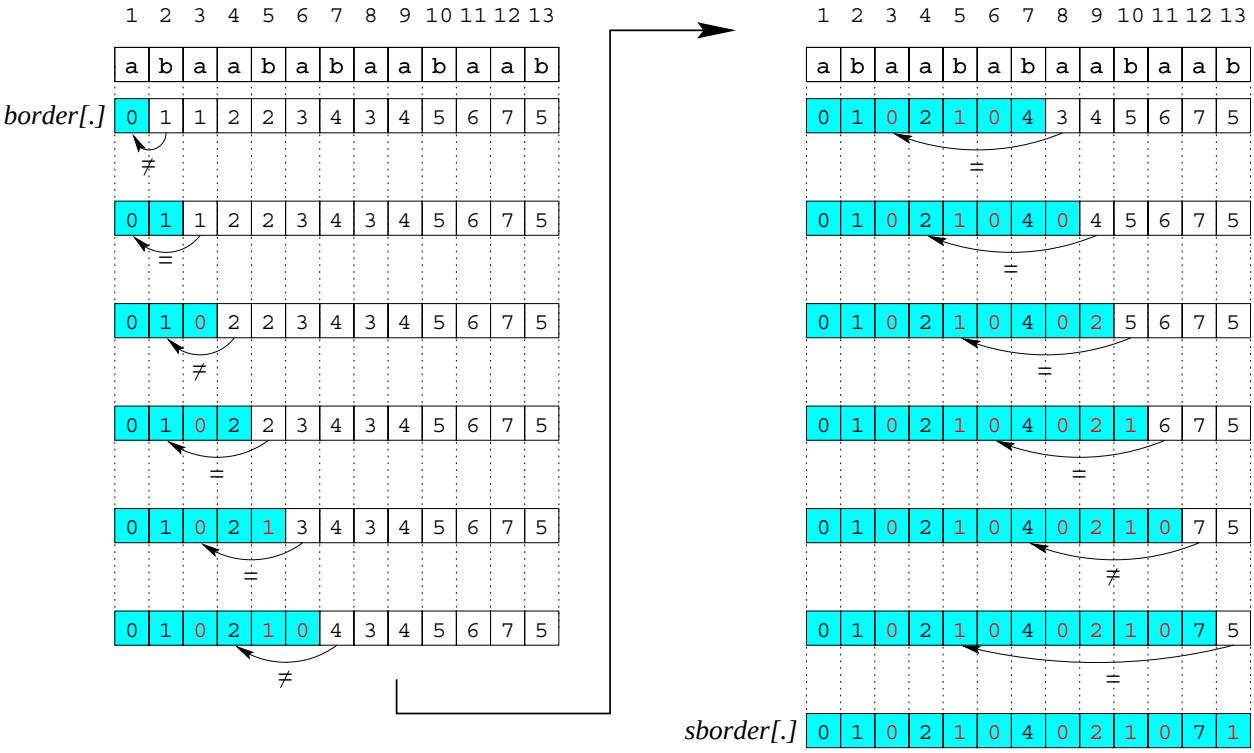
Falls kein solches k existiert, gelte $sborder[j] = 0$ (Bedeutung: im Text um 1 weiter!)

Beispiel: $sborder[j]$ lautet für F_7:

j	1	2	3	4	5	6	7	8	9	10	11	12	13
$pat[j]$	a	b	a	a	b	a	b	a	a	b	a	a	b
$border[j]$	0	1	1	2	2	3	4	3	4	5	6	7	5
$sborder[j]$	0	1	0	2	1	0	4	0	2	1	0	7	1

Wann immer $pat[border[j]] = pat[j]$ gilt, ist dort $sborder[j] < border[j]$.

Berechnung von $sborder[j]$ aus $border[j]$:



Algorithmus von Knuth, Morris und Pratt

Der Algorithmus von Knuth, Morris und Pratt ist identisch zum bisherigen Algorithmus bis auf die Tatsache, dass statt *border* die strengere Variante *sborder* verwendet wird.

Zur Berechnung von *sborder* sind ebenfalls nur marginale Änderungen notwendig:

for $j := 1$ **to** m **do** $border[j] := 0$ **end**;
$i := 2$; $j := 1$;
while $i \leq m$ **do**
  while $i + j - 1 \leq m$ **and** $pat[j] = pat[i + j - 1]$ **do**
    $j := j + 1$;
    $sborder[i + j - 1] := sborder[j]$; /* Da an der getesteten Stelle kein Unterscheid ist, muss wegen $pat[k] = pat[j]$ auf das nächstkürzere Präfix gegangen werden! */
  end
  $sborder[i + j - 1] := \max(j; sborder[i + j - 1])$; /* Da an der getesteten Stelle ein Unterscheid ist, kann wegen $pat[k] \neq pat[j]$ das Maximum genutzt werden! */
  $i := i + j - sborder[j]$;
  $j := \max(sborder[j]; 1)$
end

Der Algorithmus von Knuth, Morris und Pratt arbeitet in

- Zeit $O(n+m)$ und
- Platz $O(m)$.

Beweis:

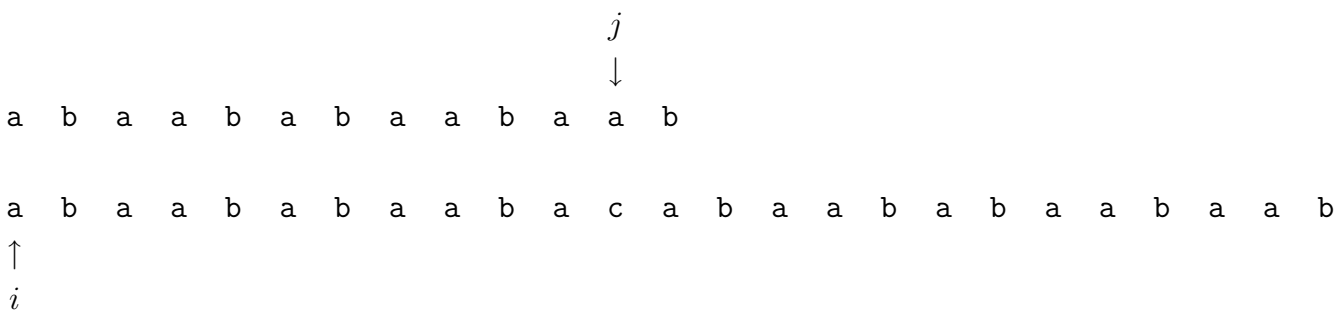
sborder kann wie *border* in Zeit $O(m)$ berechnet werden.

Durch die Verwendung von *sborder* statt *border* finden im Algorithmus nicht mehr Vergleiche statt als im Algorithmus von Morris und Pratt.

Beispiel:

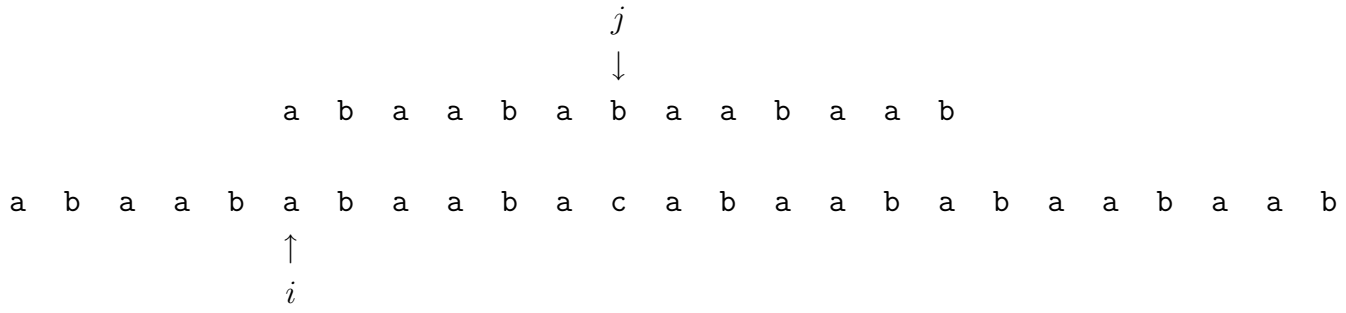
Der String F_7 wird im String `abaababaabacabaababaabaab` gesucht.

Nach dem Start des Algorithmus von Knuth, Morris und Pratt ergibt sich ein Mismatch für $j = 12$ und $i = 1$.



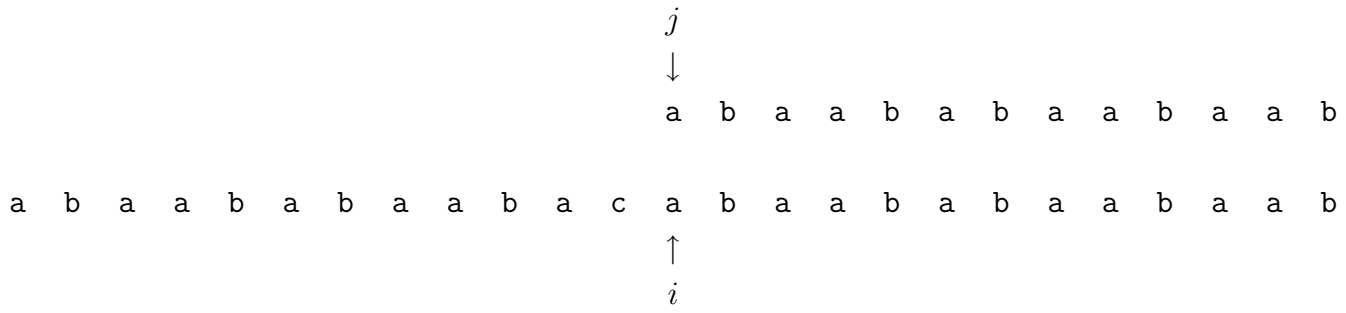
Nun erhält man

- $i := i + j - sborder[j]$ und damit $i := 1 + 12 - sborder[12] = 6$
- $j := \max(sborder[j]; 1)$ und damit $j := sborder[12] = 7$



Es tritt wieder ein Mismatch auf, deshalb $i := i + j - sborder[j] = 6 + 7 - 4 = 9$ und $j := \max(sborder[j]; 1) = \max(4; 1) = 4$.

Die Mismatches setzen sich fort bis $j = 1$ und $i = 13$.



Ab hier wird nun ein Match ermittelt.

5.1.3 Der Algorithmus von Boyer und Moore

Der Algorithmus von Boyer und Moore kann als eine verbesserte Variante eines naiven String-Matching-Algorithmus angesehen werden, bei dem *pat* mit *text* von rechts nach links verglichen wird.

Algorithmus zum naiven String-Matching von rechts nach links:

$i := 1$;
while $i \leq n - m + 1$ **do**
    $j := m$; /* Starte den Vergleich am rechten Ende von *pat* */
    while $j \geq 1$ **and** $pat[j] = text[i + j - 1]$ **do** $j := j - 1$ **end**;
    if $j = 0$ **then return true**;
    $i := i + 1$;
end
return false;

Der Algorithmus von Boyer und Moore basiert auf der folgenden Überlegung:

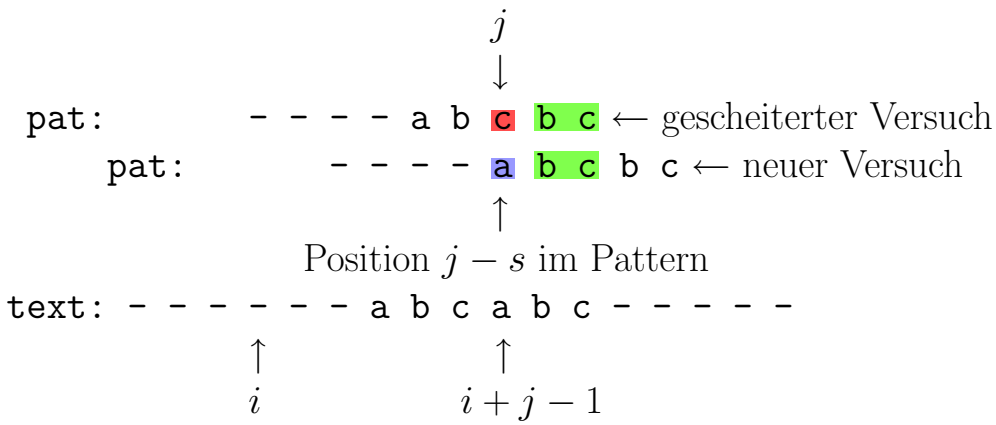
- Tritt im obigen Algorithmus an Position j ein Mismatch auf und kommt $pat[(j+1)\ldots m]$ nicht ein weiteres mal in pat als Substring vor, so kann pat gleich um m Zeichen nach rechts verschoben werden.
- Vergleicht man dagegen von links nach rechts, kann pat nach einem Mismatch an Position j nie um mehr als j Positionen nach rechts verschoben werden.

Kommt es im obigen Algorithmus an der Stelle j von pat zu einem Mismatch, so gilt $pat[(j+1)\ldots m] = text[(i+j)\ldots(i+m-1)]$ und $pat[j] \neq text[i+j-1]$.

Dies kann wie folgt ausgenutzt werden:

- Angenommen, pat tritt in $text$ an einer Position $i < i+s < i+m$ auf.
- Dann müssen die folgenden Bedingungen gelten:
- (BM1) $\forall k \in \{j+1,\ldots,m\} : k \leq s \vee pat[k-s] = pat[k]$ (d.h., für den schon geprüften Bereich)
- (BM2) $s < j \Rightarrow pat[j-s] \neq pat[j]$ (wegen des Mismatch an der Stelle $pat[j]$)

Veranschaulichung: ($m = 9$, $j = 7$, $s = 2$)



Damit man keinen Match verpasst, muss s wiederum möglichst klein gewählt werden.

Die entsprechenden Werte werden in einer Preprocessingphase ermittelt und in der Shift-Tabelle D abgelegt.

$$D[j] := \min\{s \mid s > 0 \wedge ((\text{BM1}) \text{ und } (\text{BM2}) \text{ gilt für } j \text{ und } s)\}$$

Der Algorithmus von Boyer und Moore verwendet nun im Falle eines Mismatches an Position j den in $D[j]$ abgelegten Wert, um *pat* nach rechts zu verschieben.

Algorithmus von Boyer und Moore

$i := 1$;
while $i \leq n - m + 1$ **do**
    $j := m$;
    while $j \geq 1$ **and** $pat[j] = text[i + j - 1]$ **do** $j := j - 1$ **end**;
    if $j = 0$ **then return true**;
    $i := i + D[j]$; /* Statt $i := i + 1$ beim naiven Ansatz */
end
return false;

Der Vergleich erfolgt dann wieder ab der letzten Stelle im Pattern.

Beispiel: $D[j]$ lautet für F_7:

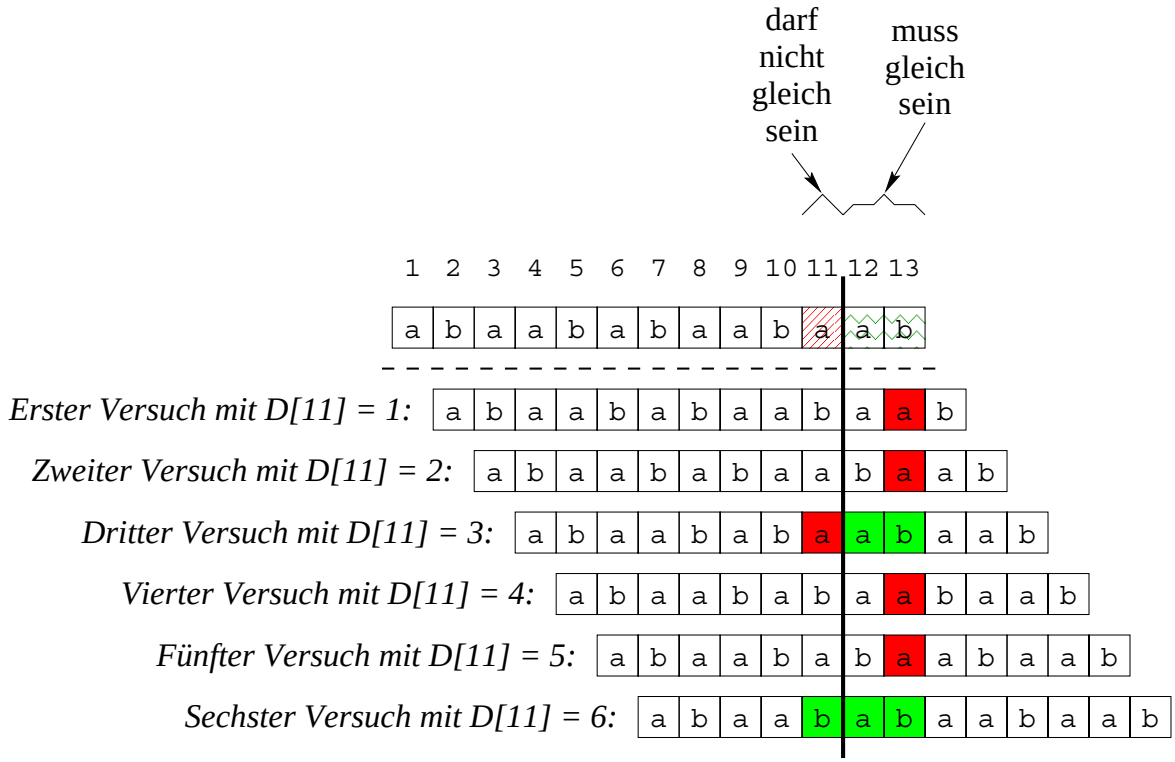
j	1	2	3	4	5	6	7	8	9	10	11	12	13
$pat[j]$	a	b	a	a	b	a	b	a	a	b	a	a	b
$D[j]$	8	8	8	8	8	8	8	3	11	11	6	13	1

- $D[13] = 1$, weil man bei einem Mismatch gleich an der letzten Stelle nur eine Position weiter kann.
- $D[12] = 13$, weil es in F_7 kein `b` ohne ein `a` davor gibt! Man kann deshalb gleich um die ganze Länge von F_7 weitergehen.
- $D[11] = 6$, weil die in F_7 am weitesten rechts stehende Folge `ab`, vor der kein `a` steht, von hinten gesehen an der Position $13 - 7 = 6$ beginnt. (Fall 1 $\rightarrow$)
- $D[10] = 11$, weil es in F_7 keine Folge `aab` gibt, vor der kein `b` steht, andererseits aber das Ende von `aab` mit dem `ab` am Anfang von F_7 übereinstimmt. (Fall 2 $\rightarrow$)

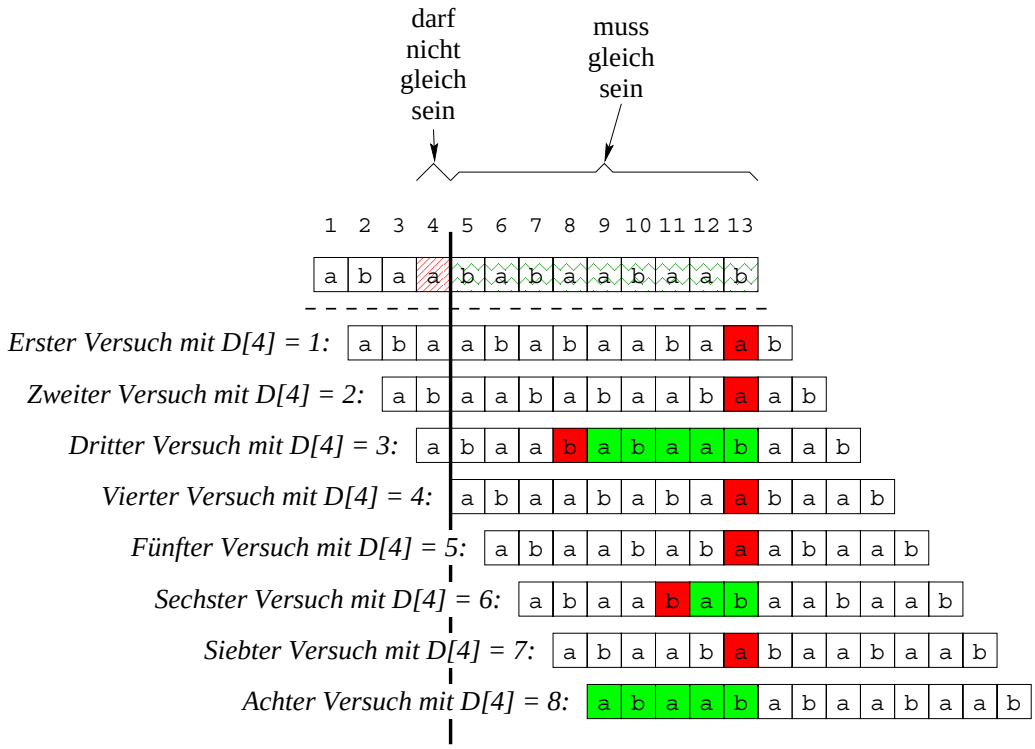
Allgemein kann man $D[j]$ wie folgt bestimmen:

Man probiert alle Werte für s beginnend mit 1 solange, bis zum ersten mal (BM1) und (BM2) für s und j gelten. Das gefundene s ist der Wert für $D[j]$.

Beispiel: Berechnung von $D[11]$ (Fall 1: $D[j] < j$)



Beispiel: Berechnung von $D[4]$ (Fall 2: $D[j] \geq j$)



Der Algorithmus von Boyer und Moore kann noch weiter verbessert werden:

Tritt an Stelle m von pat (also bereits beim ersten Vergleich) ein Mismatch auf, so wird pat bisher nur um eine Stelle nach rechts verschoben.

Es sei $last[c]$ für jedes $c \in \Sigma$ die jeweils letzte Position von c in pat.

Formal:

- $last[c] := \max_{1 \leq j \leq m} \{j \mid pat[j] = c\}$
- und $last[c] := 0$ falls c nicht in pat auftritt.

Kommt es nun an Stelle j zu einem Mismatch, kann statt

$$i := i + D[j]$$

die Anweisung

$$i := i + \max(D[j]; j - last[text[i + j - 1]])$$

verwendet werden. Damit ergeben sich noch größere Verschiebungen.

Beispiel:

Für den Suchbegriff *Datenbanksystem* ergibt sich, sofern Groß- und Kleinschreibung nicht unterschieden wird:

D	a	t	e	n	b	a	n	k	s	y	s	t	e	m
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

c	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
$last[c]$	7	6	0	1	14	0	0	0	0	0	9	0	15	8	0	0	0	0	12	13	0	0	0	0	11	0

Bemerkungen:

- Bei Einsatz der zweiten Hälfte von $i := i + \max(D[j]; j - last[text[i + j - 1]])$ wird das letzte Vorkommen des aktuellen Zeichens aus dem Text ($text[i + j - 1]$) im Pattern richtig über die Textstelle gelegt.
- Verschiebungen der Länge $j - last[text[i + j - 1]]$ heissen *Occurrence-Shift*.
- Wird nur der Occurence-Shift verwendet, d.h. die Verschiebeanweisung lautet

$$i := i + \max(1; j - last[text[i + j - 1]])$$

  so spricht man von einem vereinfachten Boyer-Moore-Algorithmus.
- Die Worst-Case-Laufzeit des vereinfachten Boyer-Moore-Algorithmus beträgt $O(nm)$.
- Auf gewöhnlichen Texten verhält sich die vereinfachte Version i.d.R. nur marginal schlechter als die ursprüngliche Version.
- Bei kleinem $|\Sigma|$ ist die Occurence-Heuristik i.d.R. weniger wirksam.

Es bleibt das Problem, die Shift-Tabelle D zu berechnen.

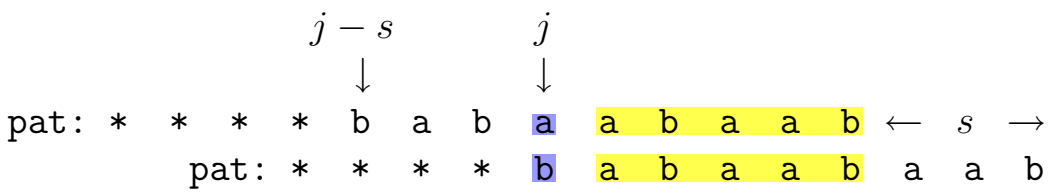
Dies geschieht in zwei Phasen.

In der ersten Phase werden nur Verschiebungen der Form $D[j] < j$ betrachtet.

Für solche Verschiebungen muss gelten:
Es gibt einen Substring $pat[(j + 1 - s) \ldots (m - s)]$ in pat, mit

- (BM1) $pat[(j + 1) \ldots m] = pat[(j + 1 - s) \ldots (m - s)]$
- (BM2) $pat[j] \neq pat[j - s]$

Veranschaulichung:



Die Situation ist ähnlich wie beim Verfahren von Morris und Pratt.

Statt eines Präfixes muss hier aber ein Suffix von *pat* mit einem inneren Teil von *pat* übereinstimmen.

Vorgehensweise für die erste Phase:

man berechnet zunächst die Hilfsstruktur $rborder[j]$ (für $0 \leq j \leq m$) mit

$$rborder[j] := \max_{1 \leq k \leq m-j} \{k \mid pat[(j+1)\ldots(j+k-1)] = pat[(m-k+2)\ldots m]\}$$

bzw.

$rborder[j] := \max_{1 \leq k \leq m-j} \{k \mid pat[(j+1)\ldots(j+k-1)]$ ist echter Suffix von $pat[(j+1)\ldots m]\}$

Weiterhin gelte $rborder[m] = 0$.

Beispiel: $rborder[j]$ lautet für F_7:

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$pat[j]$		a	b	a	a	b	a	b	a	a	b	a	a	b
$rborder[j]$	6	5	4	3	2	6	5	4	3	2	1	1	1	0

Man beachte: $rborder[j]$ entspricht $border[j]$ für pat^R.

Dementsprechend lässt sich $rborder[j]$ analog zum entsprechenden Algorithmus berechnen, wobei man nun aber von rechts nach links vorgeht.

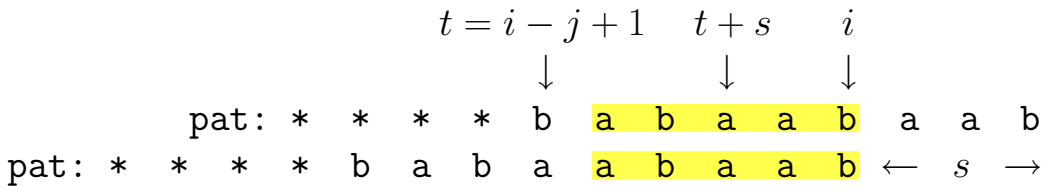
$i := m - 1;$
$j := 1;$
while $i \geq 0$ **do**
  while $i - j + 1 \geq 1$ **and** $pat[m - j + 1] = pat[i - j + 1]$ **do**
    /* Suche zur Zeichenkette von i abwärts das längste passende Suffix von pat */
    $j := j + 1;;$
    $rborder[i - j + 1] = j;$
  end
  $i := i - j + rborder[m - j + 1];$
  $j := max(rborder[m - j + 1]; 1);$
end

Wegen (BM2) ($pat[j-s] \neq pat[j]$) treten relevante Situationen zur Berechnung von $D[j]$ nur im Falle eines Mismatch in der inneren While-Schleife auf.

Dementsprechend wird zur Berechnung von $D[j]$ der Algorithmus hinter der inneren While-Schleife um die folgenden Anweisungen erweitert ($D[j]$ sei zuvor für alle j mit m initialisiert):

if $j > 1$ **then**
  $s := m - i$; /* Anzahl der Schritte, um die verschoben wurde */
  $t := i - j + 1$; /* An dieser Stelle trat die Ungleichheit auf */
  if $t + s > s$ **then** /* $t + s$ ist die Position, an der das Suffix von *pat* beginnt */
    $D[t + s] = \min(s; D[t + s])$; /* nur, wenn s kleiner ist */
  endif
endif

Veranschaulichung: ($i = 10$, $j = 6$, $s = m - i = 13 - 10 = 3$, $t = i - j + 1 = 10 - 6 + 1 = 5$)



$$\Rightarrow D[t + s] = \min(s; D[t + s]) = \min(3; D[t + s]) = 3$$

Damit steht der Algorithmus für die erste Phase.

Beispiel: $D[j]$ nach der ersten Phase für den String F_7:

j	1	2	3	4	5	6	7	8	9	10	11	12	13
$pat[j]$	a	b	a	a	b	a	b	a	a	b	a	a	b
$D[j]$	13	13	13	13	13	13	13	3	13	13	6	13	1

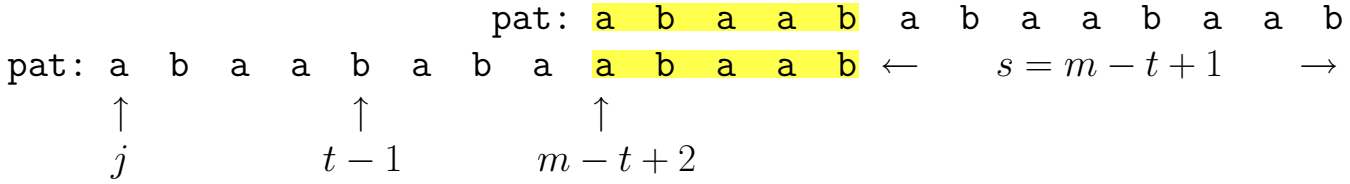
In der zweiten Phase werden Verschiebungen mit $D[j] \geq j$ betrachtet (in diesen Fällen „endet“ der Substring schon vor der Ungleichheitsstelle, d.h. er steht am Anfang von pat).

Für solche Verschiebungen muss die Bedingung

(BM1) $pat[1 \ldots (t-1)] = pat[(m-t+2) \ldots m]$ mit $j \leq m-t+1$

gelten. (BM2) ist in dieser Phase stets wahr $\rightarrow$ braucht nicht betrachtet zu werden.

Veranschaulichung: ($m = 13$, $j = 1$, $t = 6$)



Es werden nun in absteigender Reihenfolge mögliche Werte für t betrachtet, und $D[j]$ wird entsprechend korrigiert ($D[j] = m - t + 1$).

Der größte mögliche Wert für t ergibt sich durch $rborder[0]$ (Länge des längsten Substrings, der sowohl echter Präfix als auch echter Suffix ist).

Die weiteren möglichen Werte für t ergeben sich durch $rborder[m-t+1]$.

```
t := rborder[0];
l := 1;
while t > 0 do
    s = m − t + 1;
    for j := l to s do
        D[j] := min(D[j]; s);
    end
    t := rborder[s];
    l := s + 1;
end
```


Berechnung der Shift-Tabelle für Boyer und Moore

```
/* Initialisierung */
rborder[m] := 0; D[m] := 1;
for j := m − 1 downto 0 do
    rborder[j] = 1; D[j] = m;
end
/* Phase 1 */
i := m − 1;
j := 1;
while i ≥ 0 do
    while i − j + 1 ≥ 1 and pat[m − j + 1] = pat[i − j + 1] do
        j := j + 1;
        rborder[i − j + 1] = j;
    end
    if j > 1 then
        s := m − i;
        t := i − j + 1;
        if t + s > s then
```



```
            D[t + s] = min(s; D[t + s]);
        endif
    endif
    i := i − j + rborder[m − j + 1];
    j := max(rborder[m − j + 1]; 1);
end
/* Phase 2 */
t := rborder[0];
l := 1;
while t > 0 do
    s = m − t + 1;
    for j := l to s do
        D[j] := min(D[j]; s);
    end
    t := rborder[s];
    l := s + 1
end
```

Beispiel zum Algorithmus von Boyer und Moore

Der String F_7 wird in dem String `abaababaabacabaababaabaab` gesucht.

```
a b a a b a b a a b a a b
  a b a a b a b a a b a a b
              a b a a b a b a a b a a b
                a b a a b a b a a b a a b
                  a b a a b a b a a b a a b
                        a b a a b a b a a b a a b
a b a a b a b a a b a c a b a a b a b a a b a a b
```

j	1	2	3	4	5	6	7	8	9	10	11	12	13
$pat[j]$	a	b	a	a	b	a	b	a	a	b	a	a	b
$D[j]$	8	8	8	8	8	8	8	3	11	11	6	13	1

Beispiel: Shift-Tabellen für Boyer/Moore

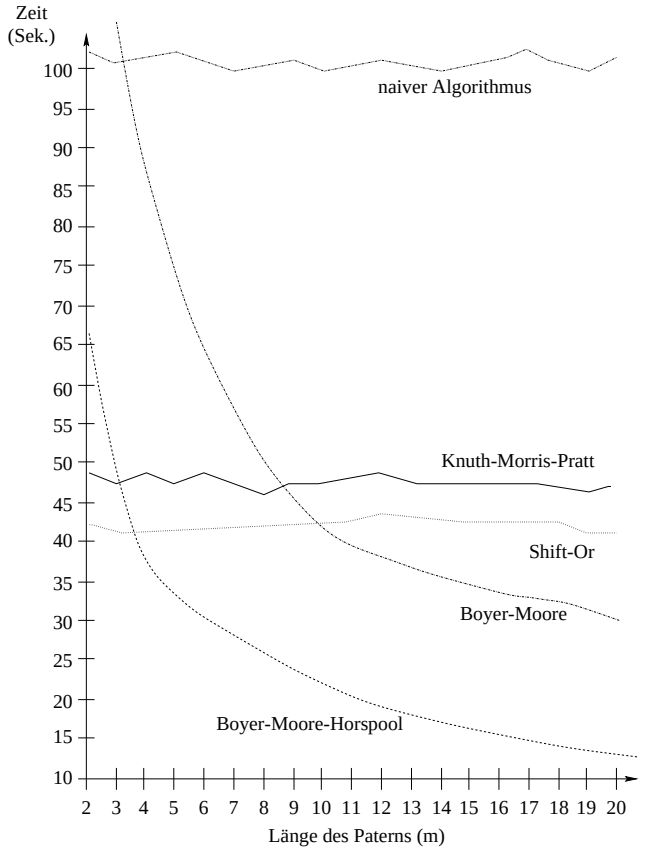
datenbank	retrieval	compiler
999999991	999999991	88888881

kuckuck	rokoko	papa	abrakadabra
3336661	662641	2241	77777771131
			00

Der Algorithmus von Boyer und Moore arbeitet in Zeit $O(n+m)$ und Platz $O(m)$.

Bemerkungen:

- Als scharfe obere Schranke für die Anzahl an Zeichenvergleichen ergibt sich $3n$.
- Würde man statt (BM1) und (BM2) nur (BM1) verwenden, so wäre keine lineare Laufzeit mehr gewährleistet ($O(mn)$).
- Sucht man mit dem Algorithmus von Boyer und Moore nach allen Matches für *pat* in *text*, so ist die Laufzeit ebenfalls $O(mn)$.



Vergleich der Verfahren anhand eines englischsprachigen Textes nach [Fuh96]

Boyer-Moore-Horspool = vereinfachter Boyer-Moore (nur mit Occurrence-Heuristik)

Shift-Or = Pattern-Matching Algorithmus, der recht effizient in Hardware realisiert werden kann

5.2 Matching von Wortmengen

Das Problem des exakten String-Matchings wird wie folgt verallgemeinert:

statt eines einzelnen Musters ist nun eine Menge von Mustern gegeben.

Problem: Matching von Wortmengen

> Gegeben sei eine Menge $P = \{pat_1; \ldots; pat_k\}$ von Strings und ein String *text*. Man bestimme, ob für ein i (mit $1 \leq i \leq k$) pat_i ein Substring von *text* ist.

Bemerkungen:

- Im folgenden bezeichnet $m := \Sigma_{i=1}^{k} |pat_i|$ die Gesamtlänge der Strings in P.
- Durch k-fache Anwendung des Algorithmus von Knuth, Morris Pratt bzw. Boyer-Moore kann das Problem in Zeit $O(m + kn)$ gelöst werden.
- Im folgenden wird gezeigt, dass das Problem in Zeit $O(m + n)$ und Platz $O(m)$ lösbar ist.

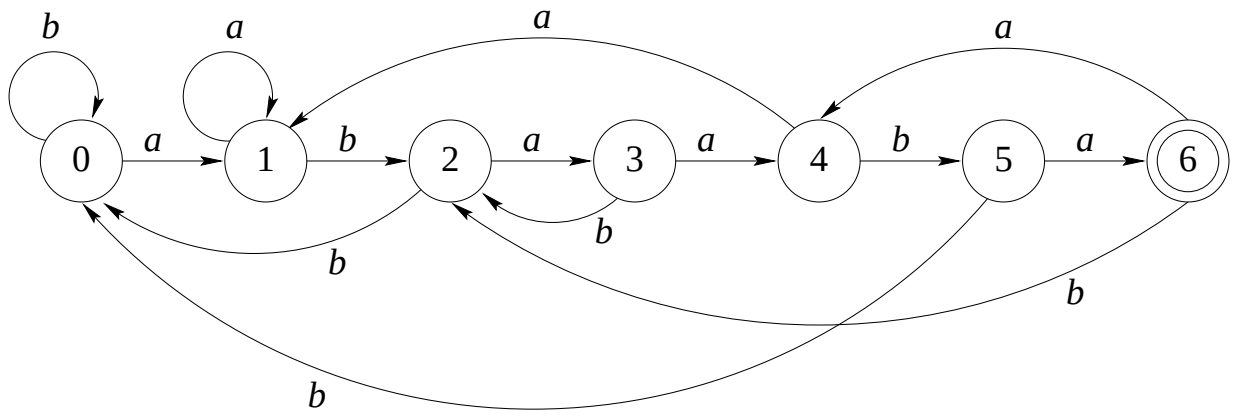
Grundidee zur Lösung:

- Man konstruiere einen deterministischen endlichen Automaten, der die Strings aus P erkennt.
- Genauer: Man baue einen Automaten für die Sprache

$$\Sigma^* \{pat_1; \ldots; pat_k\}$$

Veranschaulichung:

Für $\Sigma = \{a; b\}$ und $P = \{abaaba\}$ kann der folgende Automat verwendet werden:



Bemerkungen:

- $pat = abaaba$ ist genau dann in *text* enthalten, wenn bei Anwendung des obigen Automaten auf *text* irgendwann der Zustand 6 erreicht wird.
- Liegt der Automat für einen String *pat* vor, kann in $O(n)$ geprüft werden, ob *pat* in *text* vorkommt.
- Der zugehörige Algorithmus simuliert hierzu einfach die Abarbeitung des Automaten.
- wesentlicher Nachteil: i. Allg. hat solch ein Automat $O(|\Sigma|m)$ Zustandsübergänge.

Dieser automatenbasierte Ansatz wird nun auf eine Menge von Strings verallgemeinert.

Definition: Aho-Corasick Pattern-Matching-Automat

Ein *Aho-Corasick Pattern-Matching-Automat (AC-Automat)* besteht aus den folgenden Komponenten:

(i) Einer endlichen Menge Q von Zuständen,

(ii) einem endlichen Eingabealphabet Σ,

(iii) einer Transitionsfunktion $g : Q \times \Sigma \longrightarrow Q \cup \{fail\}$,

(iv) einer Fehlerfunktion $h : Q \longrightarrow Q \cup \{fail\}$,

(v) einem initialen Zustand $q_0 \in Q$ und

(vi) einer Menge $F \subset Q$ von Endzuständen.

Solch ein AC-Automat wird wie folgt zur Suche nach einer Menge von Mustern eingesetzt.

Algorithmus von Aho-Corasick

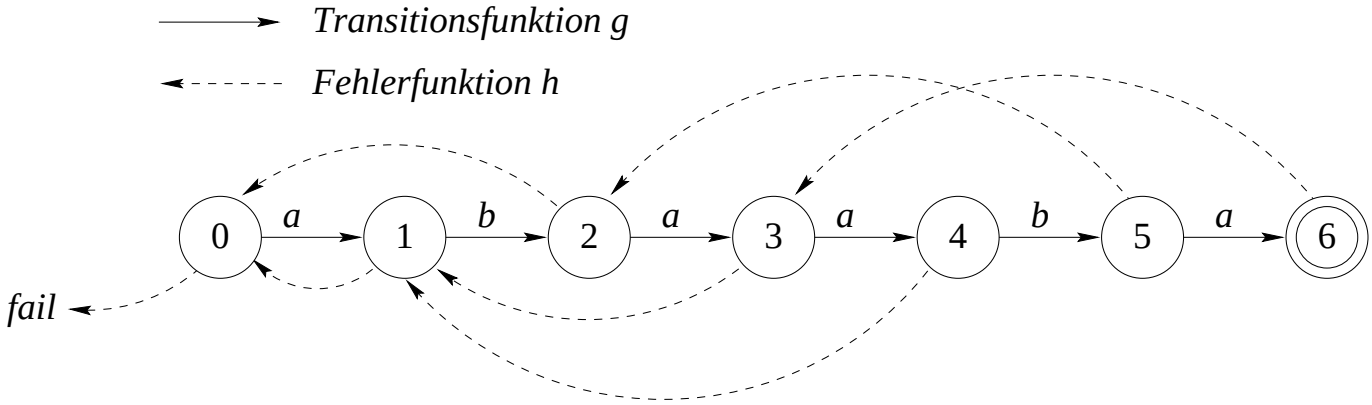
$q := q_0$;
for $i := 1$ **to** n **do**
  while $q \neq fail$ **and** $g(q; text[i]) = fail$ **do** $q := h(q)$ **end**;
  if $q = fail$ **then** $q := q_0$ **else** $q := g(q; text[i])$ **endif**;
  if $q \in F$ **then return true**;
end
return false;

Bemerkungen:

- Für jedes Symbol von *text* wird zunächst versucht, der Transitionsfunktion g zu folgen.
- Ist dies nicht möglich, wird die Fehlerfunktion h angewendet, bis ein Zustandsübergang möglich ist.
- Findet man auch mit h keinen möglichen Zustandsübergang, beginnt man wieder von vorne.

Beispiel: AC-Automat für einelementiges P

Es sei $P = \{abaaba\}$. Der AC-Automat sieht dann folgendermassen aus:



Bemerkungen:

- Die Fehlerfunktion h entspricht der Tabelle *border* aus dem Algorithmus von Morris und Pratt.
- Durch die Fehlerfunktion ist die Größe des AC-Automaten $O(m)$ und somit unabhängig von $|\Sigma|$.

Problem: den AC-Automaten für eine beliebige Menge P von Strings zu erzeugen.

Die meisten Komponenten eines AC-Automaten ergeben sich aus dem sogenannten *Trie* für die Menge P.

Definition: Trie

Es sei Σ ein endliches Alphabet. Ein *Trie* (auch Σ-Baum genannt) ist ein Wurzelbaum mit den folgenden Eigenschaften:

(i) Die Kanten des Baumes sind mit Zeichen aus Σ markiert.

(ii) Für jeden Knoten k des Baumes und jedes Zeichen $c \in \Sigma$ gibt es höchstens eine Kante, die von k ausgeht und mit c markiert ist.

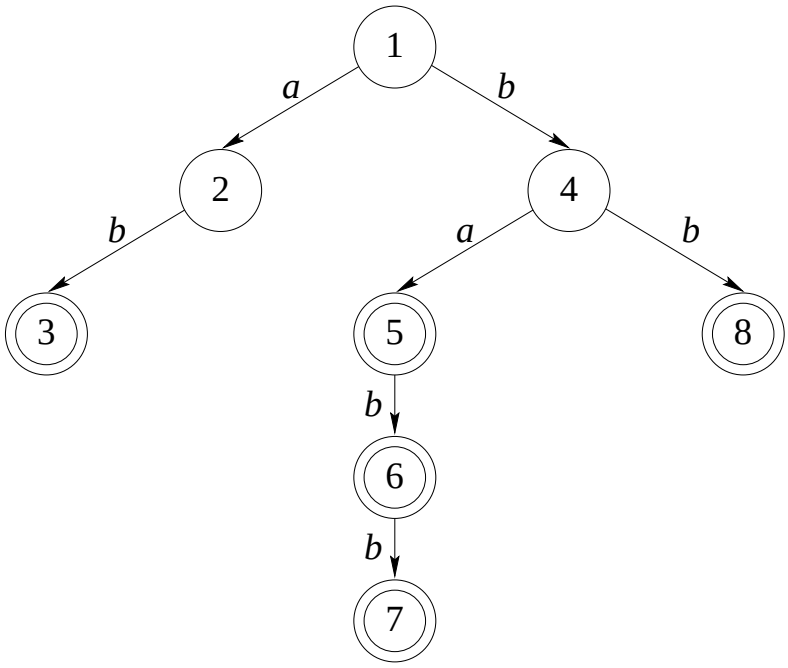
Für einen Knoten k in einem Trie bezeichnet $path(k)$ den String, der sich aus der Folge der Kantenmarkierungen auf dem Pfad von der Wurzel nach k ergibt.

Für eine Menge P von Strings bezeichnet *Trie(P)* den kleinsten Trie mit der Eigenschaft:

$$\forall pat \in P : \exists k : pat = path(k)$$

Beispiel: Trie für eine Menge von Strings

Für $P = \{ab; ba; babb; bb\}$ ergibt sich der folgende Trie:



Konsequenzen für die Konstruktion des AC-Automaten für P :

- *Trie(P)* kann in Zeit $O(m)$ konstruiert werden.
- Die Zustandsmenge Q entspricht den Knoten von *Trie(P)*.
- q_0 ist die Wurzel von *Trie(P)*.
- Die Transitionsfunktion g ergibt sich
  - aus den Kanten von *Trie(P)* und
  - der Bedingung: $g(q; c) = fail$ genau dann, wenn keine mit c markierte Kante von dem Knoten q ausgeht.
- Die Endzustände ergeben sich aus der folgenden Bedingung:
  $q \in Q$ ist genau dann ein Endzustand, wenn ein $pat \in P$ existiert, mit pat ist Suffix von $path(q)$.

Genauer: es gibt drei disjunkte Klassen von Endzuständen:

(i) Jedes Blatt von *Trie(P)* ist ein Endzustand. (Beispiel: Knoten 3)

(ii) Jeder innere Knoten q mit $path(q) \in P$ ist ein Endzustand.
Solche Endzustände treten auf, wenn ein $pat_i \in P$ echter Präfix eines $pat_j \in P$ ist.
(Beispiel: Knoten 5)

(iii) Jeder innere Knoten q, für den ein $pat \in P$ existiert mit
pat ist echter Suffix von $path(q)$ ist ein Endzustand.
Solche Endzustände treten auf, wenn ein $pat_i \in P$ Substring und nicht echter Suffix oder Präfix eines $pat_j \in P$ ist.
Anders ausgedrückt: pat_i tritt im Innern von pat_j auf. (Beispiel: Knoten 6)

Die Endzustände, die unter (i) oder (ii) fallen, können beim Aufbau von *Trie(P)* direkt erkannt werden.

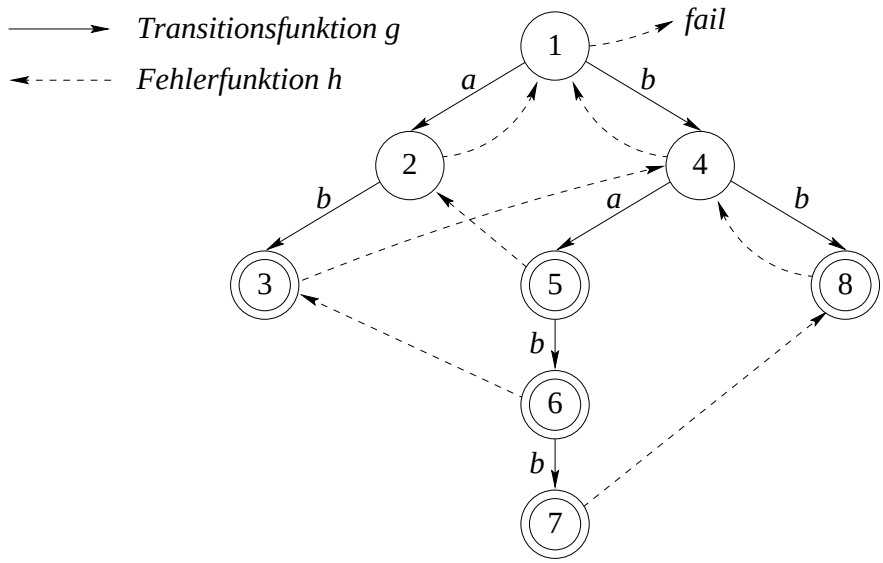
Es bleibt das **Problem**,

- die Fehlerfunktion h zu berechnen und
- Endzustände der Klasse (iii) zu erkennen.

Die Fehlerfunktion h des AC-Automaten wird wie folgt definiert:

$h(q) = q'$ gilt genau dann, wenn q' unter den Knoten von *Trie(P)* den längsten echten Suffix von *path(q)* liefert. Falls kein solches q' existiert, gelte $h(q) = fail$.

Beispiel: Fehlerfunktion eines AC-Automaten; Es sei $P = \{ab; ba; babb; bb\}$.



Bemerkungen:

- Die Fehlerfunktion stellt die Verallgemeinerung von *border* auf eine Menge von Strings dar.
- Durch die Maximalität in der Definition der Fehlerfunktion wird sichergestellt, dass kein Vorkommen eines Musters übersehen wird (entspricht der kleinsten Verschiebung).
- Die Fehlerfunktion geht im Baum stets nach oben.

Algorithmus: Berechnung der Fehlerfunktion eines AC-Automaten

$h(q_0) := fail$;
forall q mit q ist Sohn von q_0 **do** $h(q) := q_0$ **end**;
forall q' mit $depth(q') \geq 2$ in BFS-Reihenfolge **do**
    q sei der Vater von q';
    c sei das Symbol mit $g(q;c) = q'$; /* das letzte übereinstimmende Zeichen */
    $r := h(q)$; /* da wären wir vom Vater aus hin gesprungen */
    while $r \neq fail$ **and** $g(r;c) = fail$ **do**
        $r := h(r)$; /* Wir lassen uns weiter „zurückfallen“ */
    end
    if $r = fail$ **then**
        $h(q') := g(q_0;c)$;
    else
        $h(q') := g(r;c)$;
        if $h(q')$ ist Endzustand **then**
            nimm q' in die Menge der Endzustände auf;
        endif
    endif
end

Der obige Algorithmus berechnet die Fehlerfunktion eines AC-Automaten in Zeit $O(m)$.

Das Matching von Wortmengen kann mit dem Algorithmus von Aho-Corasick in Zeit $O(n+m)$ und Platz $O(m)$ gelöst werden.

Beweis:

Korrektheit

- Gemäß der Definition der Fehlerfunktion h gilt im Algorithmus von Aho-Corasick am Ende der Schleife:

$$path(q) = text[(i - depth(q) + 1) \ldots i]$$

und

$$depth(q) = max\,\{l \mid pat_j[1 \ldots l] = text[(i - l + 1) \ldots i]\ (1 \leq j \leq k)\}$$

d.h., *path(q)* entspricht dem längsten Präfix eines Patterns aus P, der mit einem Suffix des bisher gelesenen Teils von *text* (nämlich $text[1 \ldots i]$) übereinstimmt.

- Es sei i die erste Position von *text*, an der ein Vorkommen eines Strings aus P **endet**.

  Von den Strings aus P, die an i enden, sei pat_j der längste.

  $i' \leq i$ sei die Position, an der pat_j beginnt.

  q sei der Zustand des AC-Automaten nach der Bearbeitung von $text[i]$.

- Man kann zwei Fälle unterscheiden:

  (i): Es existiert $i'' < i'$ mit $text[i'' \ldots i]$ ist echter Präfix eines Patterns $pat_{j''}$ aus P. Falls mehrere i'' existieren, betrachte man deren Maximum (kleinster Wert für i'').

  Es sei $l := i - i'' + 1$ die Länge des zu $pat_{j''}$ gehörigen Präfixes.

  Aus der Invariante und der Maximalität von i'' folgt: $path(q) = pat_{j''}[1 \ldots l]$.

  Nach Konstruktion ist pat_j in $pat_{j''}$ enthalten, und somit ist q ein Endzustand der Klasse (iii).

  (ii): Gilt (i) nicht, dann ist gemäß der Invariante pat_j der längste erkannte Präfix, und q ist ein Endzustand der Klasse (i) oder (ii).

Laufzeit und Platz

- Preprocessing in $O(m)$.
- Platz wird nur für den Trie benötigt.
- Die Transitionsfunktion g wird höchstens n mal erfolgreich angewendet.
- Bei jeder erfolgreichen Anwendung von g geht man im Trie um genau eine Stufe nach unten.
- Die Fehlerfunktion h geht im Baum stets nach oben.
- Somit kann h höchstens n mal angewendet werden.

$\Rightarrow O(m+n)$

5.3 Matching regulärer Ausdrücke

Statt einer endlichen Anzahl von Mustern möchte man eventuell ein **Musterschema** vorgeben.

Zur Spezifikation von Musterschemata eignen sich **reguläre Ausdrücke**.

Problem: Matching regulärer Ausdrücke

Gegeben seien ein regulärer Ausdruck r und ein String *text*.

$L(r)$ bezeichne die durch r definierte Sprache.

Man bestimme, ob ein Substring s von *text* existiert mit $s \in L(r)$.

Bemerkungen:

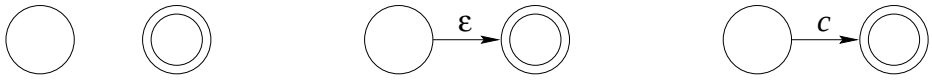
- m bezeichnet im folgenden die Länge von r.
- Durch einen regulären Ausdruck ist u.U. eine unendliche Menge von Mustern gegeben.
- Die Grundidee zum Matching regulärer Ausdrücke ist analog zum Matching von Wortmengen:
  $\rightarrow$ Man konstruiere einen Automaten zur Erkennung der Sprache $\Sigma^* L(r)$.
- Hier sind verschiedene Ansätze zur Automatenkonstruktion sinnvoll:
  (i) Konstruktion eines nichtdeterministischen Automaten oder
  (ii) eines deterministischen Automaten.
- Im folgenden wird nur Ansatz (i) näher untersucht.

Vorgehensweise:

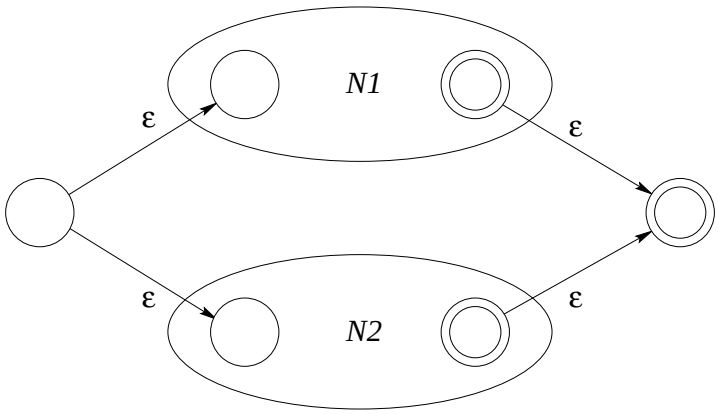
- Aus r wird ein nichtdeterministischer endlicher Automat (NEA) mit ϵ-Übergängen konstruiert.
- Ein NEA mit ϵ-Übergängen ist ein gerichteter Graph, bei dem
  - die Knoten die Zustände darstellen,
  - jede Kante mit einem Symbol aus $\Sigma \cup \{\epsilon\}$ markiert ist,
  - ein Zustand als initialer Zustand ausgezeichnet ist und
  - einige Zustände Endzustände sind.
- Ein NEA akzeptiert einen String, wenn ein Pfad vom initialen Zustand zu einem Endzustand existiert, für den die Konkatenation der beteiligten Kanten den String ergeben.
- Die Konstruktion des NEA erfolgt analog zum Beweis des Satzes über die Äquivalenz von regulären Ausdrücken und endlichen Automaten.

Die Konstruktion des Automaten basiert auf den folgenden fünf Regeln:

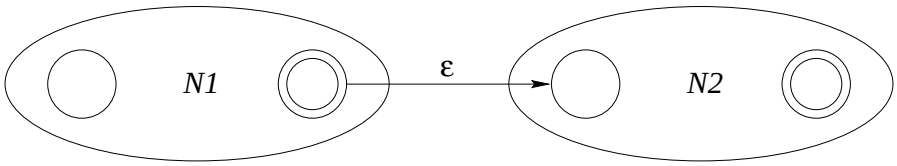
(1) Für die elementaren regulären Ausdrücke $\emptyset$; ϵ und $c \in \Sigma$ werden die folgenden Automaten konstruiert:



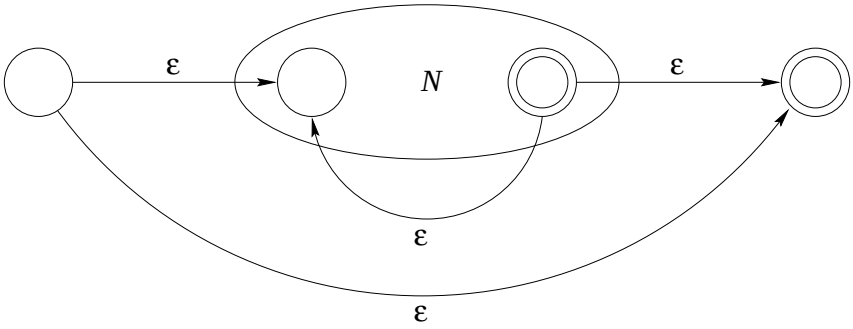
(2) Gegeben seien die regulären Ausdrücke r_1 und r_2 sowie die zugehörigen Automaten N_1 und N_2. Dann wird für den regulären Ausdruck $r_1|r_2$ (Alternativen) der folgende Automat konstruiert:



(3) Gegeben seien die regulären Ausdrucke r_1 und r_2 sowie die zugehörigen Automaten N_1 und N_2. Dann wird für den regulären Ausdruck $r_1 r_2$ der folgende Automat konstruiert:



(4) Gegeben sei der reguläre Ausdruck r sowie der zugehörige Automat N. Dann wird für r^* der folgende Automat konstruiert:



(5) Für den regulären Ausdruck (r) wird der Automat von r genutzt.

Bemerkungen:

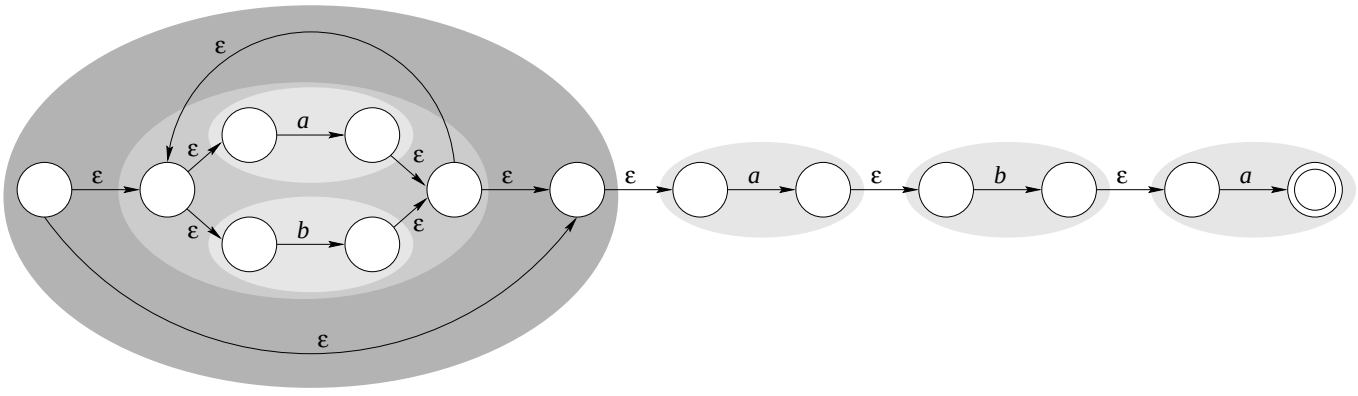
- Der nach den Regeln (1) bis (5) für r konstruierte Automat N hat höchstens $2m$ Zustände, denn jede Regel erzeugt nicht mehr als zwei neue Zustände.
- Die Regeln (1) bis (5) benötigen bei der Konstruktion jeweils nur konstante Zeit.
- N hat genau einen Start- und genau einen Endzustand.
- Der Startzustand hat keine eingehende Kante, und der Endzustand hat keine ausgehende Kante.
- Jeder Zustand hat entweder
  - genau eine ausgehende Kante, die mit einem Symbol $c \in \Sigma$ markiert ist oder
  - höchstens zwei mit ϵ markierte ausgehende Kanten.

Als Konsequenz ergibt sich:

Zu einem regulären Ausdruck r kann in $O(m)$
ein äquivalenter NEA mit ϵ-Übergängen konstruiert werden.

Beispiel: NEA für einen regulären Ausdruck

Für den regulären Ausdruck $(a|b)^*aba$ ergibt sich der folgende NEA:



Algorithmus zur Simulation eines NEA

- Im folgenden bezeichnet q_{start} den Startzustand und q_{end} den Endzustand.
- Q ist eine Menge von Zuständen.
- $g(Q; c)$ bezeichnet die Menge aller Zustände, die von einem Zustand $\in Q$ über eine mit c markierte Kante erreichbar sind.
- $eps(Q)$ bezeichnet alle Zustände, die von einem Zustand $\in Q$ über mit ϵ markierte Kanten erreichbar sind.
- Da das Muster an einer beliebigen Stelle des Textes auftreten kann, wird in jeder Iteration der Startzustand q_{start} zur aktuellen Zustandsmenge hinzugenommen.

$Q := eps(\{q_{start}\})$;
if $q_{end} \in Q$ **then return true**;
for $i := 1$ **to** n **do**
    $Q := eps(g(Q; text[i]) \cup \{q_{start}\})$;
    if $q_{end} \in Q$ **then return true**;
end
return false;

Es sei r ein regulärer Ausdruck und N der zugehörige NEA.

Dann kann mit dem obigen Algorithmus in Zeit $O(nm)$ geprüft werden, ob ein String $text$ einen auf r passenden Substring enthält.

Beweis:

- N hat $O(m)$ Zustände.
- Q kann über einen Bitvektor repräsentiert werden.
- Jeder Zustand hat höchstens zwei ausgehende Kanten.

$\Rightarrow g(Q; text[i])$ und $eps(Q)$ können in $O(m)$ berechnet werden.

Damit ergibt sich, dass das Matching regulärer Ausdrücke in Zeit $O(nm)$ und Platz $O(m)$ gelöst werden kann, weil in der Schleife n Zeichen betrachtet werden müssen.

Ein anderer Ansatz zum Matching regulärer Ausdrücke besteht darin, statt eines NEA einen deterministischen Automaten zu verwenden.

Bemerkungen:

- Ein NEA kann stets in einen äquivalenten deterministischen Automaten überführt werden (siehe *Formale Sprachen*).
- Mit einem deterministischen Automaten könnte die Erkennung eines Musters in Zeit $O(n)$ erfolgen (vgl. Matching von Wortmengen).
- **Problem:** Die Größe eines deterministischen Automaten ist im worst case exponentiell in m.

Mit einem deterministischen Automaten kann das Matching regulärer Ausdrücke in Zeit $O(2^m + n)$ und Platz $O(2^m)$ gelöst werden.

5.4 Approximatives String-Matching

- Bisher waren die Probleme derart, dass das Muster *pat* exakt mit einem Substring von *text* übereinstimmen musste.
- Beim Matching von regulären Ausdrücken ließ man zwar Varianten zu, aber ebenfalls keine Fehler.
- In vielen praktischen Fällen ist es wünschenswert, die Stellen von *text* zu finden, die mit *pat* „nahezu“ übereinstimmen.
- Anwendungsbeispiele:
  - Molekularbiologie (Erkennung von DNA-Sequenzen)
  - Ausgleich verschiedener Schreibweisen (Grafik vs. Graphik)
  - Ausgleich von Beugungen
  - Toleranz gegenüber Tippfehlern
  - Toleranz gegenüber OCR-Fehlern
- Der Begriff „nahezu“ wird dabei durch eine Metrik auf Strings formalisiert.

- Zur Erinnerung:

  Sei M eine Menge.

  Eine Funktion $d : M \times M \rightarrow \mathbb{R}$ heißt Metrik, wenn die folgenden Bedingungen erfüllt sind:

  - $d(x; y) \geq 0$ für alle $x; y \in M$
  - $d(x; y) = 0 \Leftrightarrow x = y$ für alle $x; y \in M$
  - $d(x; y) = d(y; x)$ für alle $x; y \in M$
  - $d(x; z) \leq d(x; y) + d(y; z)$ für alle $x; y; z \in M$.

- $(M; d)$ ist dann ein metrischer Raum.

Wir definieren das Problem des approximativen String-Matching damit wie folgt:

Gegeben seien ein String *pat*, ein String *text*, eine Metrik d für Strings und ein ganze Zahl $k \geq 0$. Man finde alle Substrings y von *text* mit $d(pat; y) \leq k$.

Bemerkungen:

- Für $k = 0$ erhält man Problem alle Vorkommen eines exakt vorgegebenen Strings in einem Text zu suchen.
- Das Problem des approximativen String-Matching ist zunächst ein „abstraktes“ Problem, da nichts über die Metrik d ausgesagt wird.
- Zur Konkretisierung des Problems und zur Entwicklung von entsprechenden Algorithmen müssen zunächst sinnvolle Metriken betrachtet werden.

Definition: Hamming-Distanz

Für zwei Strings x und y mit $|x| = |y| = m$ ergibt sich die Hamming-Distanz durch:

$$d(x, y) = |\{i \mid 1 \leq i \leq m \wedge x[i] \neq y[i]\}|$$

Bemerkungen:

- Die Hamming-Distanz ist die Anzahl der Positionen, an denen sich x und y unterscheiden.
- Die Hamming-Distanz ist nur für Strings gleicher Länge definiert.
- Wird beim Problem des approximativen String-Matching für d die Hamming-Distanz verwendet, so spricht man auch von „string matching with k mismatches“ .

Beispiel: Die Hamming-Distanz der Strings *abcabb* und *cbacba* beträgt 4.

Definition: Editierdistanz, Levenstein-Metrik

- Für zwei Strings x und y ist die Editierdistanz $edit(x;y)$ definiert als die kleinste Anzahl an Einfüge- und Löschoperationen, die notwendig sind, um x in y zu überführen.
- Lässt man zusätzlich auch die Ersetzung eines Symbols zu, so spricht man von einer Levenstein-Metrik $lev(x;y)$.
- Nimmt man als weitere Operation die Transposition (Vertauschung zweier benachbarter Symbole) hinzu, so erhält man die Damerau-Levenstein-Metrik $dlev(x;y)$.

Bemerkungen:

- Offensichtlich gilt stets $dlev(x;y) \leq lev(x;y) \leq edit(x;y)$.
- Die Damerau-Levenstein-Metrik wurde speziell zur Tippfehlerkorrektur entworfen.
- Wird beim Problem des approximativen String-Matching für d eine der obigen Metriken verwendet, dann spricht man auch von „string matching with k differences“ bzw. von „string matching with k errors“.

Beispiel:

Für $x = abcabba$ und $y = cbabac$ gilt:

$edit(x; y) = 5$

$$abcabba \rightarrow bcabba \rightarrow cabba \rightarrow cbba \rightarrow cbaba \rightarrow cbabac$$

$dlev(x; y) = lev(x; y) = 4$

$$abcabba \rightarrow cbcabba \rightarrow cbabba \rightarrow cbaba \rightarrow cbabac$$

$$abcabba \rightarrow bcabba \rightarrow cbabba \rightarrow cbabab \rightarrow cbabac$$

Problem: Berechnung der Stringdistanz

- Gegeben seien zwei Strings x und y.
- Man ermittle $edit(x; y)$ bzw. $lev(x; y)$ bzw. $dlev(x; y)$
- sowie die zugehörigen Operationen zur Überführung der Strings.

Bemerkungen:

- Wenn x und y Dateien repräsentieren, wobei $x[i]$ bzw. $y[j]$ die i-te Zeile bzw. j-te Zeile darstellt, dann spricht man auch vom File Difference Problem.
- Unter UNIX steht das Kommando `diff` zur Lösung des File Difference Problems zur Verfügung.
- Da die Metriken *edit*, *lev* und *dlev* sehr ähnlich sind, wird im Folgenden nur die Levenstein-Metrik betrachtet.
- Algorithmen für die anderen Metriken erhält man durch einfache Modifikationen der folgenden Verfahren.

- Im folgenden sei $m = |x|$ und $n = |y|$ und es gelte $m \leq n$.

$\rightarrow$ Lösungsansatz: dynamische Programmierung

$\rightarrow$ genauer: berechne die Distanz der Teilstrings $x[1 \ldots i]$ und $y[1 \ldots j]$ auf der Basis bereits berechneter Distanzen.

Die Tabelle LEV für die Zeichenketten x und y sei definiert durch:

$$LEV[i;j] := lev(x[1 \ldots i]; y[1 \ldots j]) \text{ mit } 0 \leq i \leq m; 0 \leq j \leq n$$

Die Werte für $LEV[i;j]$ können mit Hilfe der folgenden Rekursionsformeln berechnet werden:

- $LEV[0;j] = j$ für $0 \leq j \leq n$
- $LEV[i;0] = i$ für $0 \leq i \leq m$
- $LEV[i;j] = \min\{LEV[i-1;j]+1,$
  $LEV[i;j-1]+1,$
  $LEV[i-1;j-1]+\delta(x[i];y[j])\}$
- $\delta(a;b) = \begin{cases} 0 & \text{falls} \quad a = b \\ 1 & \text{sonst} \end{cases}$

Bemerkungen:

- Die Rekursionsformel spiegelt die drei Operationen Löschen, Einfügen und Substitution wider.
- Die Stringdistanz ergibt sich als $LEV[m;n]$.
- Möchte man nur die Stringdistanz berechnen, so genügt es, sich auf Stufe i der Rekursion die Werte von LEV der Stufe $i-1$ zu merken.
- Benötigt man die zugehörigen Operationen, speichert man LEV als Matrix und ermittelt die zugehörigen Operationen in einer „Rückwärtsrechnung“.

Algorithmus: Berechnung der Stringdistanz

for $i := 0$ **to** m **do** $LEV[i;0] := i$ **end**;
for $j := 1$ **to** n **do** $LEV[0;j] := j$ **end**;
for $i := 1$ **to** m do
  for $j := 1$ **to** n do
    $LEV[i;j] := \min\{\ LEV[i-1;j]+1;$
    $LEV[i;j-1]+1;$
    $LEV[i-1;j-1]+\delta(x[i];y[j])\};$
  end
end
return $LEV[m;n]$;

Beispiel:

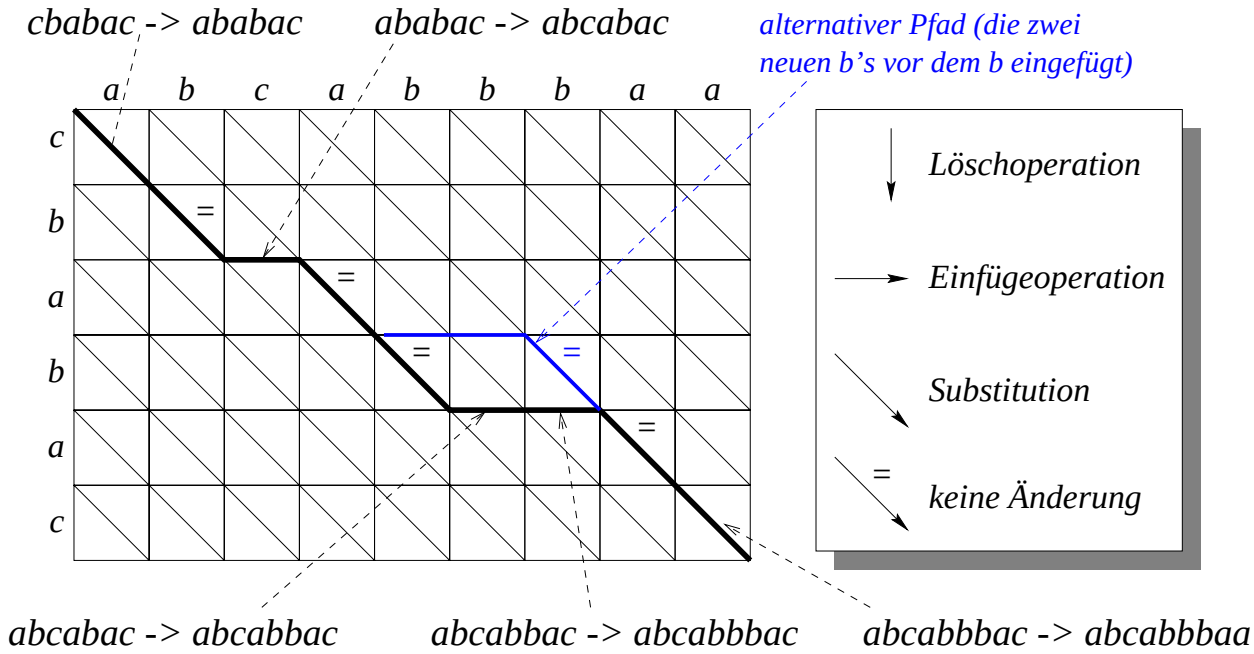
Darstellung von LEV als Matrix für $x = cbabac$ und $y = abcabbbaa$:

		a	b	c	a	b	b	b	a	a
	0	1	2	3	4	5	6	7	8	9
c	1	1	2	2	3	4	5	6	7	8
b	2	2	1	2	3	3	4	5	6	7
a	3	2	2	2	2	3	4	5	5	6
b	4	3	2	3	3	2	3	4	5	6
a	5	4	3	3	3	3	3	4	4	5
c	6	5	4	3	4	4	4	4	5	5

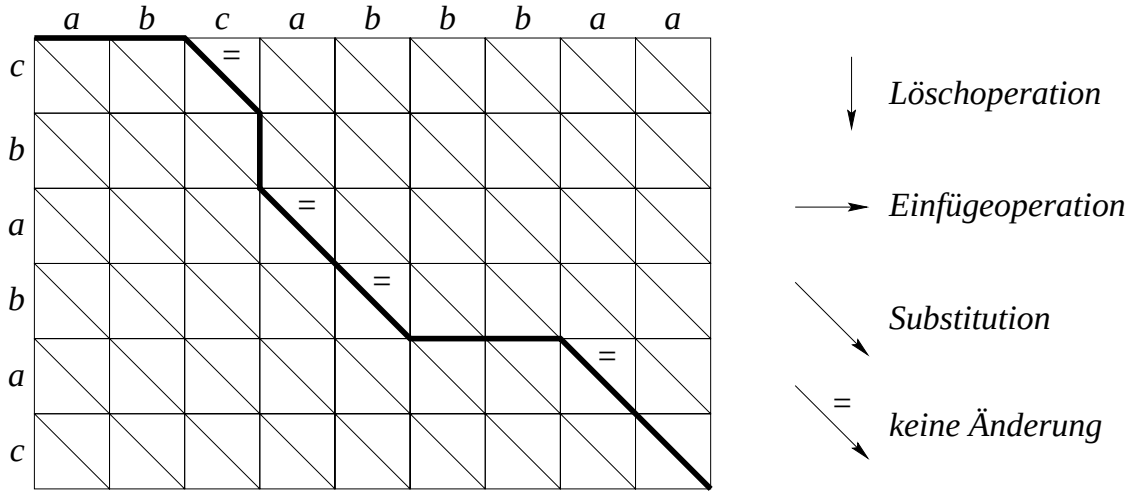
Die zugehörigen Umwandlungen lauten:

$cbabac \rightarrow ababac \rightarrow abcabac \rightarrow abcabbac \rightarrow abcabbbac \rightarrow abcabbbaa$

Veranschaulichung: Die Berechnung der Stringdistanz kann als Pfad in einem Graphen veranschaulicht werden.



Eine andere, nicht optimale Ableitung:



Der dargestellte Pfad entspricht der folgenden (nicht optimalen) Umwandlung:

$cbabac \rightarrow acbabac \rightarrow abcbabac \rightarrow abcabac \rightarrow abcabbac \rightarrow abcabbbac \rightarrow abcabbbaa$

Aus der Rekursionsformel und den Bemerkungen folgt:

- Die Stringdistanz (für *edit*, *lev* und *dlev*) kann in Zeit $O(mn)$ berechnet werden.
- Das Problem der Berechnung der Stringdistanz kann mit Platz $O(mn)$ gelöst werden.

Mit einer kleinen Änderung kann die angegebene Rekursionsformel auch zur Lösung des Problems des approximativen String-Matching eingesetzt werden.

Es sei $MLEV$ definiert durch: $MLEV[i;j] := \min_{1 \leq l \leq j} \{lev(pat[1 \ldots i]; text[l \ldots j])\}$

d.h., $MLEV[i;j]$ = kleinste Distanz zwischen $pat[1 \ldots i]$ und einem Suffix von $text[1;j]$.

Es gilt nun: $MLEV[0;j] = 0$ für $0 \leq j \leq n$,

denn $pat[1 \ldots 0] = \epsilon$ und ϵ ist stets in $text[1 \ldots j]$ ohne Fehler enthalten.

Ansonsten berechnet sich $MLEV[i;j]$ wie $LEV[i;j]$, d.h.:

- $MLEV[i;0] = i$ für $0 \leq i \leq m$
- $MLEV[i;j] = \min\{MLEV[i-1;j] + 1,$
  $MLEV[i;j-1] + 1,$
  $MLEV[i-1;j-1] + \delta(x[i]; y[j])\}$

Gilt nun $MLEV[m;j] \leq k$, so endet in Position j ein Substring y von $text$ mit $lev(pat; y) \leq k$ (wobei m die Patternlänge ist).

Beispiel: Die Tabelle $MLEV$ für $pat = ABCDE$ und $text = ACEABPCQDEABCR$.

	j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
i			A	C	E	A	B	P	C	Q	D	E	A	B	C	R
0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	A	1	0	1	1	0	1	1	1	1	1	1	0	1	1	1
2	B	2	1	1	2	1	0	1	2	2	2	2	1	0	1	2
3	C	3	2	1	2	2	1	1	1	2	3	3	2	1	0	1
4	D	4	3	2	2	3	2	2	2	2	2	3	3	2	1	1
5	E	5	4	3	2	3	3	3	3	3	3	2	3	3	2	2

Für $k = 2$ ergeben sich die Positionen 3, 10, 13 und 14.

Die zugehörigen Substrings von $text$ sind ACE, $ABPCQDE$, ABC und $ABCR$.

Das Problem des approximativen String-Matching kann für die Metriken $edit$, lev und $dlev$ in Zeit $O(mn)$ gelöst werden.

5.4.1 String-Ähnlichkeit mit Hilfe von n-grams

Eine andere Möglichkeit, die Ähnlichkeit von Zeichenketten zu bestimmen, arbeitet mit n-grams.

Ein n-gram ist dabei zunächst nichts anderes, als ein Substring der Länge n.

Beispiel: für $n = 3$ und den Begriff *Eisenbahn* ergeben sich folgende *Trigrams*:

eis, ise, sen, enb, nba, bah, ahn

für $n = 2$ ergeben sich folgende *Bigrams*:

ei, is, se, en, nb, ba, ah, hn

Gerade bei höheren Werten für n nimmt man oft zuätzlich auch ein Begrenzerzeichen auf:

#ei, eis, ise, sen, enb, nba, bah, ahn, hn#

Ähnlichkeitssuche mit n-grams

- Man kann nun bei einer Suche alle Wörter akzeptieren, deren n-gram-Zerlegung
  - gleich oder
  - bis auf einen gewissen Fehler gleich ist.

Beispiel:

Suche alle Dokumente, die ein Wort enthalten, dass bis auf maximal ein Trigram alle Trigrams aus „Eisenbahn“ enthält.

Anmerkungen:

- Oft wird mit Hilfe von n-grams auch ein sprachabhängiges Stemming ersetzt [Fra92].
- Man kann auch beim Vektorraummodell statt Wörtern n-grams als Dimensionen verwenden.

N-Grams vs. Words as Indexing Terms [MM97]

Experimente von James Mayfield und Paul McNamee, The John Hokins University, Applied Physics Laboratory, Laurel, MD, USA

Vergleich dreier „Termarten“:

1. Wörter werden als Terme verwendet (keine Stammformreduktion o.ä.)
2. Stammformen werden als Terme verwendet
3. 5-grams werden als Terme verwendet
   Dazu wird der Text wie folgt vorbereitet:
   - Entfernen der Interpunktion
   - Umstellung auf Kleinbuchstaben
   - Zahlen werden auf ein einziges Zeichen abgebildet

Zwei Arten von Termgewichtungsfunktionen wurden verwendet:

- Die „klassische“ TF/IDF-Formel (die Vorkommenshäufigkeit des „Terms“ im Dokument wird multipliziert mit dem logarithmierten Quotient aus Anzahl der Dokumente insgesamt und Anzahl der Dokumente, die den Term enthalten)
  Als Ähnlichkeitsmaß wird dann das Cosinusmaß verwendet.
- Ein Ansatz, bei dem der Zentroid aller Dokumente von jedem einzelnen Vektor abgezogen wird.
  Dadurch wird der Mittelpunkt des Koordinatensystems in die Mitte der Dokumentenkollektion verschoben.

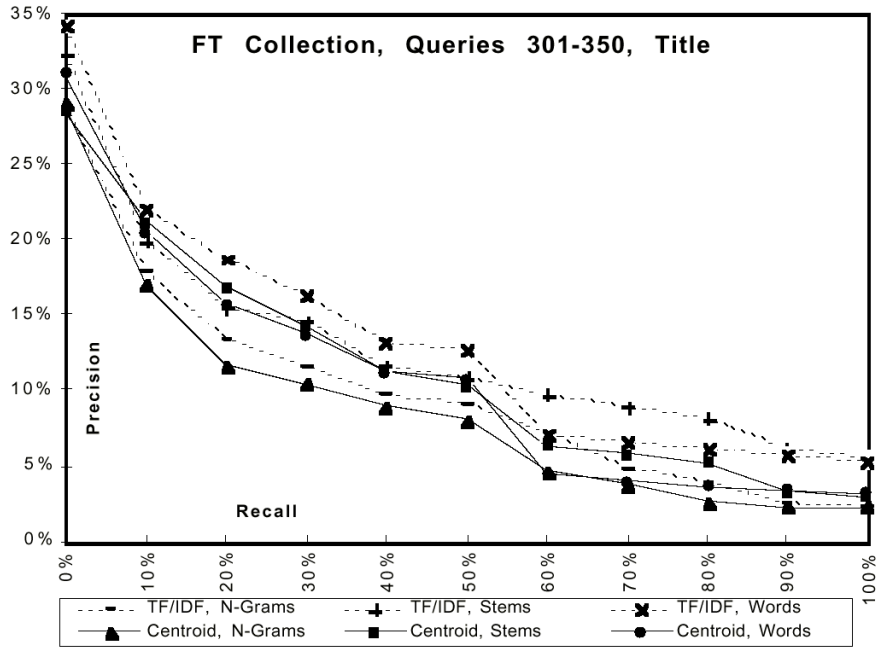
Die Experimente wurden auf der Basis der Financial Times Artikel aus der TREC-Kollektion durchgeführt.

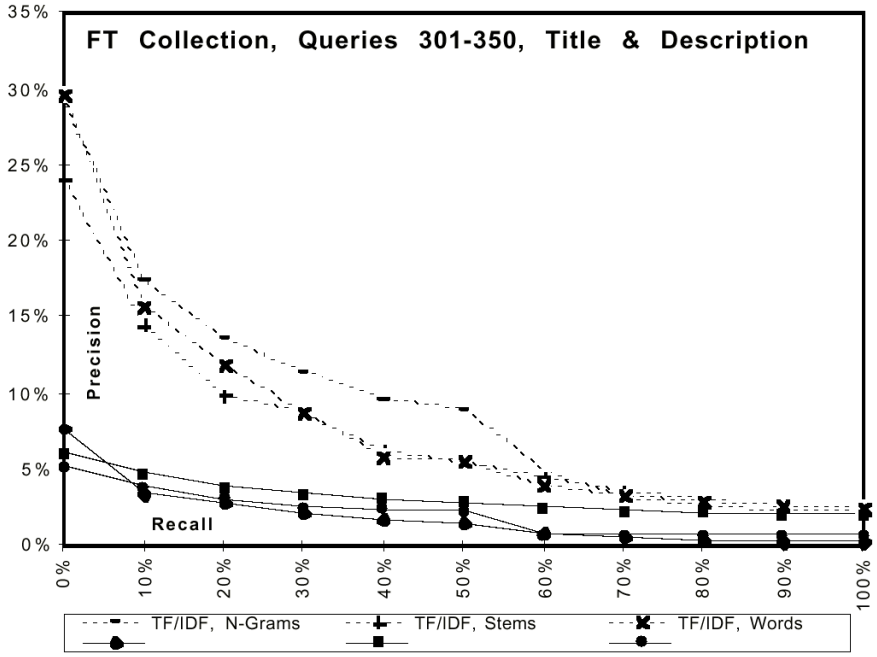
	Size (megabytes)	# Docs	Median # Words/Doc	Mean # Words/Doc
the Financial Times, 1991-1994 (FT)	564	210158	316	412,7

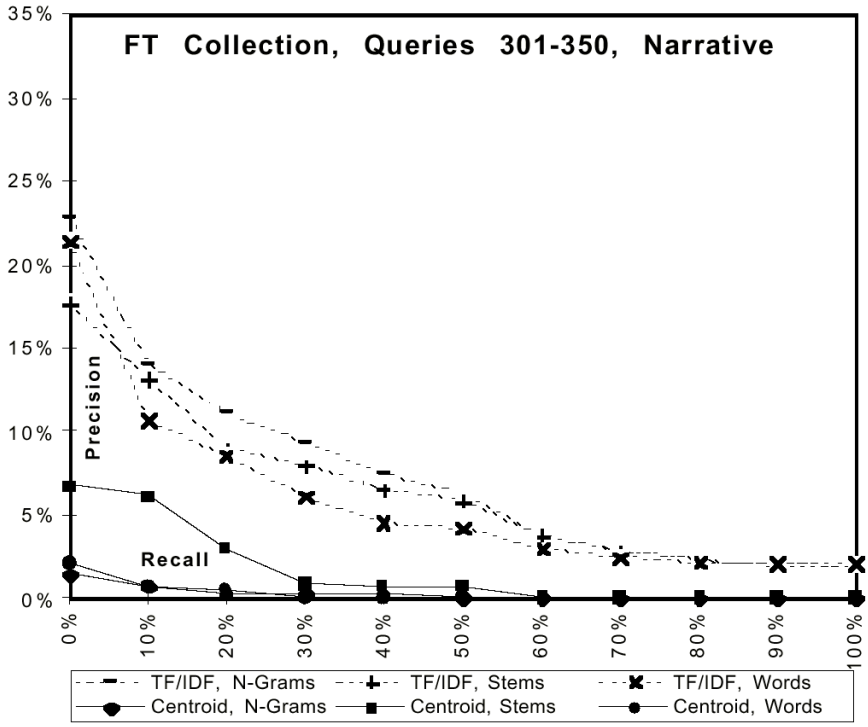
Umfang der *topic*-Definitionen in Wörten:

	Min	Max	Mean
TREC-6 (301-350)	47	156	88.4
title	1	5	2.7
description	5	62	20.4
narrative	17	142	65.3

Basis der Anfrage ist nur der Titel des *Topics*:







Schlußfolgerungen

1. Bei einem TF/IDF-Ansatz sind für kurze Anfragen Wörter oder Stammformen günstiger als 5-grams.
   Für lange Anfragen dagegen sind 5-grams besser.
2. Die klassische TF/IDF-Formel ist dem Zentroid-Ansatz überlegen.

Ein Problem bei der Verwendung von n-grams:

- Warum wird das Dokument für wichtig gehalten?
- Man kann wohl kaum die n-grams hervorheben, die zu einem hohen Ähnlichkeitswert beigetragen haben!

5.4.2 Phonetisch ähnliche Zeichenketten

- Problem: Suche nach phonetisch gleichen Wörtern
- Wörter werden hierzu auf interne Codes abgebildet.
- Phonetisch ähnliche Wörter sollen auf möglichst gleiche Codes abgebildet werden.
- Das bekannteste Verfahren ist der SOUNDEX-Algorithmus.

SOUNDEX-Algorithmus

Der SOUNDEX-Algorithmus besteht aus den folgenden Schritten:

1. Nimm den ersten Buchstaben des Wortes und transformiere die restlichen Buchstaben (unabhängig von Groß- und Kleinschreibung) nach der untenstehenden Tabelle.
2. Streiche alle Nullen.
3. Reduziere alle hintereinander vorkommenden gleichen Zahlen auf eine Zahl.
4. Beschränke den ganzen Code auf maximal vier Stellen.

Buchstaben	Code
a e i o u h w y	0
b f p v	1
c g j k q s x z	2
d t	3
l	4
m n	5
r	6

Beispiel: Die Namen *Neumann* und *Newman* werden als gleich erkannt:

Neumann $\xrightarrow{1.}$ N005055 $\xrightarrow{2.}$ N555 $\xrightarrow{3.}$ N5

Newman $\xrightarrow{1.}$ N00505 $\xrightarrow{2.}$ N55 $\xrightarrow{3.}$ N5

Bemerkungen:

- Der SOUNDEX-Algorithmus dient insbesondere für die Suche nach Namen.
- Der Code ist nicht immer erfolgreich.
- Ähnlich geschriebene Wörter werden oft auf unterschiedliche Codes abgebildet.
  Beispiel: Rodgers und Rogers.
  Rodgers $\xrightarrow{1.}$ R032062 $\xrightarrow{2.}$ R3262 $\xrightarrow{4.}$ R326
  Rogers $\xrightarrow{1.}$ R02062 $\xrightarrow{2.}$ R2262
- In der Praxis wird der SOUNDEX-Algorithmus z.T. in leicht abgeänderter Form eingesetzt.

Kapitel 6

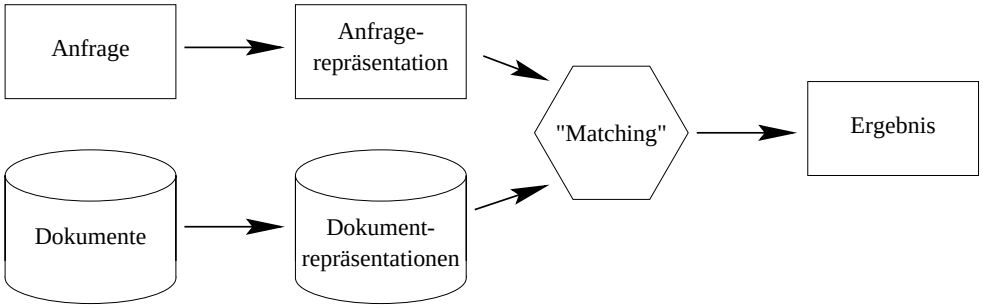
Einfache IR Modelle und ihre Implementierung

- Wortsuche
- Boolesches Retrieval
- Coordination-Level Match
- Fuzzy-Retrieval

6.1 IR Modelle

Ein IR Modell muss implizit oder explizit folgende Aspekte abdecken:

- Es muss die Annahmen angeben, die dem Modell zugrundeliegen
  Z.B. Unabhängigkeitsannahmen, ...
- Es muss eine Repräsentation für die Dokumente vorgeben.
- Es muss eine Repräsentation für die Anfragen vorgeben.
- Es muss eine „Retrieval-Funktion“ definieren, die für ein Paar aus Anfrage und Dokument eine Relevanzkennzahl berechnet.



IR Modelle nach [BYRN99]

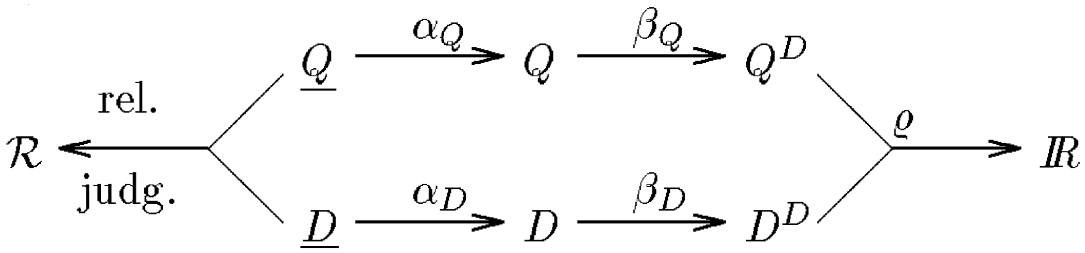
Nach Baeza-Yates besteht ein IR-Modell aus:

- einer Menge von Annahmen und
- einem Algorithmus zum „Ranken“ der Dokumente im Hinblick auf eine Benutzeranfrage.

Formal kann ein IR Modell damit als Quadruple $[\mathbf{D}, \mathbf{Q}, \mathcal{F}, R(q_i, d_j)]$ betrachtet werden:

- $\mathbf{D}$ ist eine Menge logischer Sichten (oder Repräsentationen) auf die Dokumente,
- $\mathbf{Q}$ ist eine Menge logischer Sichten (oder Repräsentationen) der Informationsbedürfnisse der Benutzer.
  Diese Repräsentationen werden als Anfragen bezeichnet.
- $\mathcal{F}$ ist eine Umgebung für die Modellierung der Dokumente und Anfragen und
- $R(q_i, d_j)$ ist eine Rankingfunktion, die zu einem Paar (q_i, d_j) einen numerischen Rankingwert berechnet.

Konzeptionelles Modell für Textretrieval nach [FB91, Fuh96]:



- $\underline{D}$ steht für die Menge der Dokumente in der Datenbasis.
- $\underline{Q}$ steht für die Menge der Anfragen an das IRS.
- Zwischen den Dokumenten und den Anfragen besteht die Relevanzbeziehung, die als Abbildung in die Menge $\mathcal{R}$ der möglichen Relevanzurteile aufgefasst wird.
- Die in dem IRS repräsentierte semantische Sicht von Dokumenten wird als Menge von Dokumentrepräsentationen D bezeichnet.
- Die formalisierten Anfragen werden als Frage-Repräsentationen Q bezeichnet.
- Q und D entstehen aus $\underline{Q}$ und $\underline{Q}$ durch die Abbildungen α_Q und α_D.

- Eine Dokumentrepräsentation kann z.B. eine Menge von Terms mit zugehörigen Vorkommenshäufigkeiten sein, eine Frage-Repräsentation ein boolescher Ausdruck mit Terms als Operanden.
- Die Repräsentationen werden für die Zwecke des Retrieval in
  - Dokumentbeschreibungen (Objektattribute) D^D und
  - Fragebeschreibungen (logische Frageformulierung) Q^D

  überführt.
- Die Retrievalfunktion ϱ vergleicht für Frage-Dokument-Paare diese Beschreibungen und berechnet daraus das Retrievalgewicht, das i.a. eine reelle Zahl ist.

6.2 Wortsuche

Gesucht werden hier alle Dokumente, die eine bestimmte Zeichenkette als Wort enthalten:

- Gegeben: Ein bestimmtes Wort W
- Gesucht: Alle Dokumente, die W enthalten

Mögliche Erweiterungen:

- Verwendung einer Stamm- oder Grundformreduktion

  Dabei werden alle Dokumente gesucht, die ein Wort enthalten, dessen Stamm- oder Grundform der Stamm- oder Grundform von W entspricht.
- Ausweitung auf Wortmengen $W_1, \ldots, W_k$

  Dabei werden alle Dokumente gesucht, die alle gegebenen Worte enthalten (UND-Verknüpfung).

Während wir beim String-Matching geprüft haben, ob ein String in einem Text enthalten ist, wird nun in einer Menge von Dokumenten nach den Dokumenten gesucht, die den String enthalten.

Mögliche Vorgehensweise:

- man betrachtet nacheinander alle Dokumente und
- prüft dabei für jedes Dokument mit den String Matching Algorithmen aus Abschnitt 5 ob das Dokument das Pattern enthält.

Problem: der Zeitaufwand wächst linear mit dem Umfang der Dokumentenkollektion!

Lösungsansatz: Um die Suche effizienter zu gestalten wird vorab eine Indexstruktur erstellt, in der gesucht wird.

6.2.1 Signaturen

Signaturen sind Bitstrings fester Länge, auf die Wörter oder Texte abbildet werden.

Definition: Signatur

Es sei $F \in \mathbb{N}$. Eine Signatur der Länge F ist ein Bitstring $s = b_1 b_2 \ldots b_F$, d.h. $b_i \in \{0; 1\}$ für $1 \leq i \leq F$.

Bemerkungen:

- Signaturen werden durch die Abbildung von Wörtern auf Bitstrings erzeugt.
- Diese Abbildung geschieht typischerweise mit Hilfe einer Hash-Funktion.
  Beispiel für eine Hashfunktion für ein l-stelliges Wort $w_1, w_2, \ldots, w_l$:

  $hash := 0;$
  for $i := 1$ **to** l **do**
  $\quad hash := hash * 157 + w_i;$
  end
  $hash := hash \bmod 2^F;$

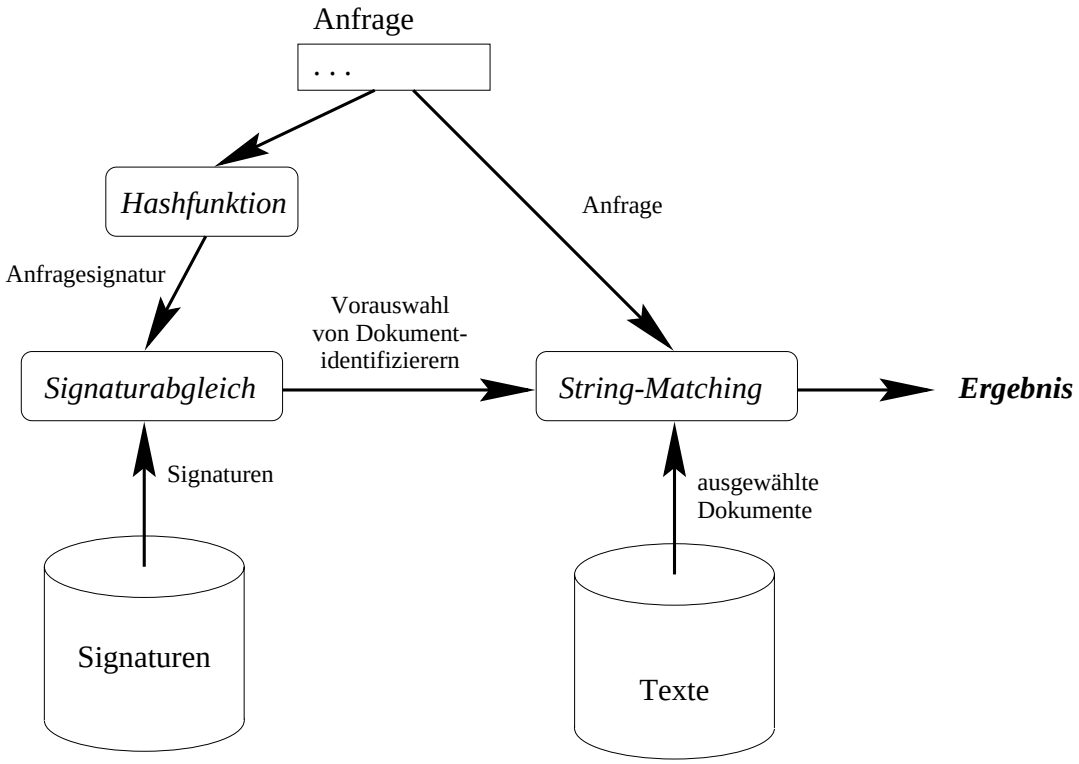
- Bei der Abbildung können künstliche Homonyme entstehen, d.h. verschiedene Wörter werden auf die gleiche Signatur abgebildet.
- Indem man die Suchoperationen auf den Signaturen der Dokumente durchführt, erhält man eine Effizienzsteigerung gegenüber der Stringsuche in den orginären Dokumenten.
- Durch spezielle Datenstrukturen ist eine weitere Beschleunigung möglich.

Grundlegende Idee von Signature-Files:

Signature-Files basieren auf der Idee des inexakten Filters, d.h.:

- Mit ihnen kann auf sehr effiziente Weise eine notwendige Bedingung überprüft werden, die von allen Dokumenten erfüllt werden muss, die ein zu suchendes Muster enthalten.
- Es kann vorkommen, dass einige Dokumente, die das zu suchende Muster nicht enthalten, ebenfalls die Testbedingung erfüllen.
  Solche Dokumente heissen false drops.
- Die meisten Dokumente, die das Suchmuster nicht enthalten, werden aber durch die Testbedingung herausgefiltert.

Bild: Architektur eines Systems mit Signatur-Files



- Die Signaturen der Dokumente werden in einer separaten Datei, dem Signature-File, verwaltet.
- Liegt eine Anfrage vor, so wird das Signature-File (i.d.R. sequentiell) durchsucht, und die meisten Dokumente werden anhand der Testbedingung verworfen.
- Die restlichen Dokumente werden mit Hilfe von String-Matching Verfahren auf Basis des eigentlichen Dokumentes überprüft und dann
  - ebenfalls verworfen (*false drops*)
  - oder als Antwort zurückgegeben.

Arten von Signaturen:

Binärsignaturen

- Bei Binärsignaturen werden die Wörter auf alle 2^F möglichen Signaturen abgebildet.
- Für eine Anfrage vergleicht man die Anfragesignatur mit den einzelnen Wortsignaturen.
- Der Effizienzgewinn entsteht dadurch, dass ein Vergleich zweier Signaturen (Anfragesignatur $\leftrightarrow$ Wortsignatur) effizienter ist als ein Vergleich auf Zeichenketten.

Überlagerungsfähige Signaturen

- Für Signature-Files werden typischerweise überlagerungsfähige Signaturen eingesetzt.
- Hierbei werden in jeder Signatur genau m Bits gesetzt.
- Somit stehen $\binom{F}{m}$ verschiedene Signaturen zur Verfügung.
- Der Parameter m wird als Signaturgewicht bezeichnet.
- Der Vorteil überlagerungsfähiger Signaturen besteht darin, dass man neben einzelnen Wörtern auch ganze Texte auf eine Signatur abbilden kann (superimposed coding).

6.2.2 Überlagerungsfähige Signaturen

Superimposed Coding:

- Jedes zu indexierende Wort wird auf einen Bitstring der Länge F abgebildet, in dem genau m Bits gesetzt sind.
- Angenommen, ein Block b besteht aus D verschiedenen Wörtern.
  Dann wird die Blocksignatur für b gebildet, indem die D Wortsignaturen mittels OR verknüpft werden.

Beispiel: Superimposed Coding

Es gilt: $F = 12$; $m = 4$ und $D = 2$.

Wort	Signatur
free	001 000 110 010
text	000 010 101 001
Blocksignatur	001 010 111 011

Hashfunktionen für Superimposed Coding

Zur Bestimmung der Signatur für ein Wort W_j verwendet man m unabhängige Hashfunktionen $h_1(W_j), \ldots, h_m(W_j)$ (nach [Knu73]).

Dabei bilden alle m Hashfunktionen h_i aus der Menge der Wörter auf eine natürliche Zahl im Bereich von 1 bis F ab.

b_x ist dann in der Signatur für das Wort W_j gleich 1, wenn es ein $i \in \{1, \ldots m\}$ mit $h_i(W_j) = x$ gibt.

Da mehrere der m Hashfunktionen den gleichen Wert liefern können, können in der Signatur auch weniger als m Bits gesetzt sein. Lösungsmöglichkeiten:

- Für $m \ll F$ kann dieser Effekt aber vernachlässigt werden.
- Man verwendet mehr als m Hashfunktionen und wendet diese der Reihe nach an, bis die Menge der Ergebniswerte m Elemente enthält.

Wie erzeugt man die m Hashfunktionen?

Es gibt viele Möglichkeiten.

Eine (etwas aufwendige) wäre in der schon bekannten Hashfunktion unterschiedliche Primzahlen p_i zu verwenden.

Für ein l-stelliges Wort $w_1, w_2, \ldots, w_l$ ergibt sich:

$hash_i := 0;$
for $k := 1$ **to** l **do**
    $hash_i := hash_i * p_i + w_k;$
end
$hash_i := hash_i \bmod F;$

Einsatz von *superimposed coding* in Signature-Files:

- F, m und D sind Entwurfsparameter.
- Dokumente werden in logische Blöcke unterteilt.
  Ein logischer Block ist ein Teil eines Dokuments, der genau D **verschiedene zu indexierende Wörter** enthält.
- Durch superimposed coding ergibt sich die Signatur des logischen Blocks.
- Die Suche nach allen Dokumenten, die das Wort w enthalten, geschieht wie folgt:
  - Für das zu suchende Wort wird die Signatur S_w berechnet.
  - Das Signature-File (bestehend aus den Blocksignaturen S_{b_i}) wird durchsucht.
    Für jede Blocksignatur wird eine (bitweise) AND Verknüpfung mit der Signatur des Suchworts durchgeführt.
  - Gilt
    $$S_w = S_w \wedge S_{b_i},$$
    d.h. alle in S_w gesetzten Bits sind auch in S_{b_i} gesetzt, dann wird das zu b_i gehörende Dokument in die Kandidatenmenge aufgenommen.

  - Um die false drops zu eliminieren, wird in den Dokumenten der Kandidatenmenge eine Stringsuche nach dem Wort w durchgeführt.

- Bei einer konjunktiven Anfrage, d.h. mehreren mit AND verknüpften Suchwörtern, kann man ähnlich vorgehen.

  Man bildet hierzu einfach durch Überlagerung die gemeinsame Signatur der einzelnen Anfragewörter.

- Eine Alternative ist, für jedes Anfragewort eine Kandidatenmenge anzulegen und diese Mengen vor dem String-Matching zu schneiden.
- Dieser Ansatz kann analog auch für beliebige boolsche Anfragen angewendet werden.

False Drops:

- Wie bereits erwähnt, können bei Signature-Files sogenannte false drops auftreten, d.h.
  - ein Block (bzw. dessen Signatur) erfüllt die notwendige Bedingung,
  - das zu suchende Muster ist aber nicht in dem Block (Dokument) enthalten.
- Die false drops können durch die Hash-Funktion oder die Überlagerung der Signaturen entstehen.

Beispiel: false drop

Es gelte: $F = 12$; $m = 3$ und $D = 3$.

Betrachte die folgenden Wortsignaturen:

text	010 010 001 000
search	010 000 100 100
method	100 100 000 001
full	010 110 000 000
knowledge	000 111 000 000
retrieval	000 000 111 000
lexicon	100 010 010 000
Anfrage:	
text search	010 010 101 100
Blocksignaturen:	
text search method	110 110 101 101
search knowledge retrieval	010 111 111 100
full text search	010 110 101 100
lexicon knowledge retrieval	100 111 111 000

Als Antwort zu obiger Anfrage enthält man die ersten drei logischen Blöcke.

Der zweite ist dabei ein false drop.

Eine wichtige Aufgabe bei der Konstruktion von Signature-Files besteht nun darin, die Parameter so zu wählen, dass möglichst wenig false drops auftreten.

Fehlerwahrscheinlichkeit

Die Parameter F, m und D sollten so gewählt werden, dass die Wahrscheinlichkeit für das Auftreten von *false drops* möglichst klein ist.

Definition: Fehlerwahrscheinlichkeit (false drop probability)

Die Fehlerwahrscheinlichkeit (false drop probability) F_d ist die Wahrscheinlichkeit dafür, dass

- eine (Block)signatur die notwendige Bedingung erfüllt, aber
- der Block das zu suchende Muster nicht enthält.

Formal:

$$F_d = P(\text{Signatur erfüllt Bedingung} \mid \text{Suchmuster nicht in Block enthalten})$$

Notationen:

F_d Fehlerwahrscheinlichkeit

F Signaturlänge

m Signaturgewicht (Anzahl gesetzter Bits)

D Anzahl der Signaturen, die überlagert werden

Voraussetzungen:

- Es werden nur einfache Anfragen betrachtet.
- Gleichverteilung der Bits auf die Signaturen.

Zunächst wird die Wahrscheinlichkeit berechnet, mit der ein Bit in einer Blocksignatur gesetzt ist.

- Die Wahrscheinlichkeit, dass ein Bit in einer Wortsignatur nicht gesetzt ist, beträgt $1 - \frac{m}{F}$.
- Für die Blocksignatur werden D Wortsignaturen überlagert.
  Damit beträgt die Wahrscheinlichkeit, dass ein
  Bit in einer Blocksignatur nicht gesetzt ist, $\left(1 - \frac{m}{F}\right)^D$.
- Die Wahrscheinlichkeit, dass ein Bit in einer Blocksignatur gesetzt ist, beträgt somit $1 - \left(1 - \frac{m}{F}\right)^D$.

Nun wird die Wahrscheinlichkeit berechnet, mit der sich eine Blocksignatur für eine einfache Anfrage qualifiziert.

- Bei einer einfachen Anfrage sind in der Anfragesignatur m Bits gesetzt.
- Eine Blocksignatur qualifiziert sich für eine Anfragesignatur, wenn alle m Bits, die in der Anfragesignatur gesetzt sind, auch in der Blocksignatur gesetzt sind.
- Diese Wahrscheinlichkeit beträgt somit:

$$\left(1 - \left(1 - \frac{m}{F}\right)^D\right)^m$$

Man geht nun sehr pessimistisch vor und geht davon aus, dass alle Blocksignaturen, die sich qualifizieren, „false drops“ sind. Also:

$$F_d = \left(1 - \left(1 - \frac{m}{F}\right)^D\right)^m$$

Bemerkungen:

- Bei steigendem F (Signaturlänge) fällt F_d monoton.
- Bei steigendem D (Überlagerungsfaktor) steigt F_d monoton.
- Für steigendes m kann F_d fallen oder steigen.
- Optimierungsproblem bei der Signaturkonstruktion:
  Gegeben sind F und D. Man bestimme m, so dass die Fehlerwahrscheinlichkeit minimiert wird.

Für große D kann F_d durch

$$F_d = \left(1 - e^{-\frac{mD}{F}}\right)^m$$

approximiert werden.

Hieraus ergibt sich der optimale Wert für m als

$$m_{opt} = \frac{F \cdot \ln 2}{D}$$

Die Fehlerwahrscheinlichkeit beträgt dann:

$$F_d = \left(\frac{1}{2}\right)^{\frac{F \cdot \ln 2}{D}} = \frac{1}{2^{\frac{F \cdot \ln 2}{D}}}$$

Weitere Konsequenz:

- Die Wahrscheinlichkeit, dass ein Bit in einer Blocksignatur gesetzt ist, beträgt für m_{opt} ungefähr $\frac{1}{2}$.

Was bedeutet dass für die Fehlerwahrscheinlichkeit? (nach [BYRN99])

- Wir können die Fehlerwahrscheinlichkeit aus dem Speicherplatz, den wir für die Signaturen „spendieren“, berechnen.
- Annahme: *ein Wort bestehe im Durchschnitt aus* c_w *Zeichen (von je 8 Bit)*
  Ferner sei Y die Gesamtzahl der Wörter in den Dokumenten

⇒ Speicherplatzbedarf für die Dokumente in Bits $S_D = Y \cdot c_w \cdot 8$
Speicherplatzbedarf für die Signaturen in Bits $S_S = Y \cdot \frac{F}{D}$
Wenn wir nun den Zusatzspeicher, den wir für die Signaturen verwenden wollen, in Relation zu S_D definieren, dann setzen wir $S_S = \alpha \cdot S_D$.
α könnten wir z.B., auf 10 % oder 20 % festlegen.
Aus α können wir dann die Werte für die anderen Parameter ableiten:

$$\begin{aligned} S_S &= \alpha \cdot S_D \\ Y \cdot \frac{F}{D} &= \alpha \cdot Y \cdot c_w \cdot 8 \\ \frac{F}{D} &= \alpha \cdot c_w \cdot 8 \\ \frac{F}{D \cdot c_w \cdot 8} &= \alpha \end{aligned}$$

- Wenn wir α vorgeben, ergibt sich $\frac{F}{D}$ wie folgt: $\frac{F}{D} = \alpha \cdot c_w \cdot 8$
- Sind wir nun bei einem Wert von $c_w = 10$ bereit 10 % zusätzlichen Speicher für Signaturen zu verwenden, ergibt sich: $\frac{F}{D} = \alpha \cdot c_w \cdot 8 = 0,1 \cdot 10 \cdot 8 = 8$
- Wenn wir dies in die Formel für die Fehlerwahrscheinlichkeit einsetzen, bedeutet dies: $F_d = \frac{1}{2^{\frac{F}{D} \cdot \ln 2}} = \frac{1}{2^{8 \cdot \ln 2}} = 2,14\%$
- Daraus folgt, dass 2,14 % der Textddokumente (unnötigerweise) sequentiell mit String-Matchung Techniken durchsucht werden müssen.
- Würden wir 20 % zusätzlichen Speicher bereitstellen, ergibt sich: $F_d = \frac{1}{2^{\frac{F}{D} \cdot \ln 2}} = \frac{1}{2^{16 \cdot \ln 2}} = 0,0459\%$

Auswirkungen von konjunktiven Anfragen

Wenn wir davon ausgehen, dass in der Anfrage A Wörter konjunktiv verknüpft werden, dann ist der Erwartungswert für die Anzahl der gesetzten Bits in der überlagerten Anfragesignatur

$$F \cdot \left(1 - \left(1 - \frac{m}{F}\right)^A\right)$$

Für die Fehlerwahrscheinlichkeit erhalten wir damit statt $F_d = \left(1 - \left(1 - \frac{m}{F}\right)^D\right)^m$ nun:

$$F_d = \left(1 - \left(1 - \frac{m}{F}\right)^D\right)^{F \cdot \left(1 - \left(1 - \frac{m}{F}\right)^A\right)}$$

Damit fällt die Fehlerwahrscheinlichkeit mit wachsendem A exponentiell!

Wenn man bereit ist, mit der gleichen Fehlerwahrscheinlichkeit zu arbeiten, könnte man somit für höhere Werte von A mit wesentlich kürzeren Signaturen arbeiten.

Beispiel:

Es seien $D = 10$, $F = 80$ und $A = 1$ (also nur ein Begriff in der Anfrage).

Dann erhalten wir

- $m_{opt} = \frac{F \cdot \ln 2}{D} = 5,545$
- $F_d = \left(1 - \left(1 - \frac{m}{F}\right)^D\right)^m = 2,45\%$

Wenn wir nun statt dessen $A = 2$ annehmen, erhalten wir mit $D = 10$, $F = 80$ und $m = 5,545$

- $F_d = \left(1 - \left(1 - \frac{m}{F}\right)^D\right)^{F \cdot \left(1 - \left(1 - \frac{m}{F}\right)^A\right)} = \left(1 - \left(1 - \frac{m}{F}\right)^D\right)^{10,706} = 0,077\%$

Wir könnten damit für $A = 2$ F auf 42 zurücknehmen und hätten immer noch eine Fehlerwahrscheinlichkeit von nur $2,33\%$.

Hätten wir dann aber in einer Anfrage statt zwei konjunktiv verknüpften Begriffen nur einen, läge die Fehlerwahrscheinlichkeit bei $14,28\%$!

Ein weiteres Problem bei konjunktiv verknüpften Anfragen ist, dass die Bearbeitung mit Signaturen einer blockweisen Betrachtung entspricht.

Folge: Es werden nur Dokumente gefunden, bei denen in einem Block alle Begriffe vorkommen!

Wendet man Signaturen z.B. im Zusammenhang mit der Suche nach Mehrwortgruppen an, kann man sich behelfen:

- geht man davon aus, dass die Teile einer Mehrwortgruppe in einem Bereich von j Worten vorkommen müssen, und
- sei ferner $D \gg j$,
- dann kann man die hintereinander liegenden Blöcke jeweils um j Wörter überlappen lassen.

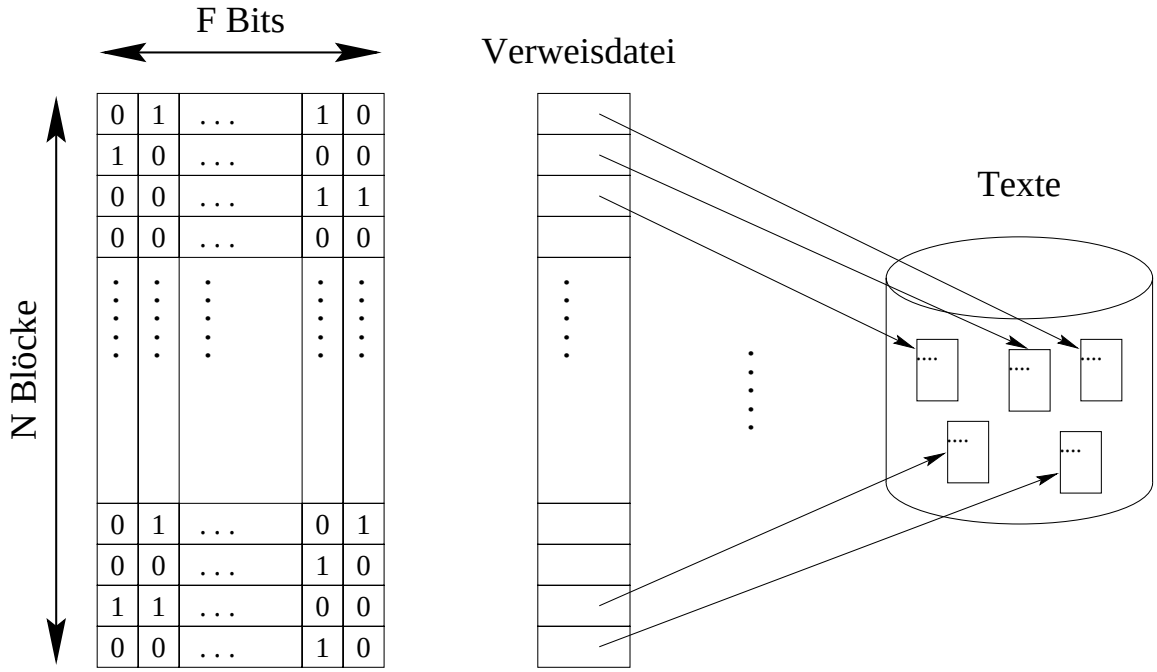
6.2.3 Speicherungsstrukturen für Signature-Files

Sequentielles Signature-File:

- Im Prinzip stellt ein Signature-File für N Blöcke eine Bitmatrix mit F Spalten und N Zeilen dar ($N \times F$-Matrix).
- Die einfachste Möglichkeit besteht darin, die einzelnen Zeilen der Matrix sequentiell zu speichern.
- Liegt diese Organisationsform vor, spricht man von einem sequentiellen Signature-File (Sequential Signature File, SSF).
- Hierbei wird zusätzlich eine Datei mit Verweisen auf die Dokumente (oder Blöcke) eingeführt.

  Alternativ könnten diese Verweise auch mit in das Signature-File aufgenommen werden.

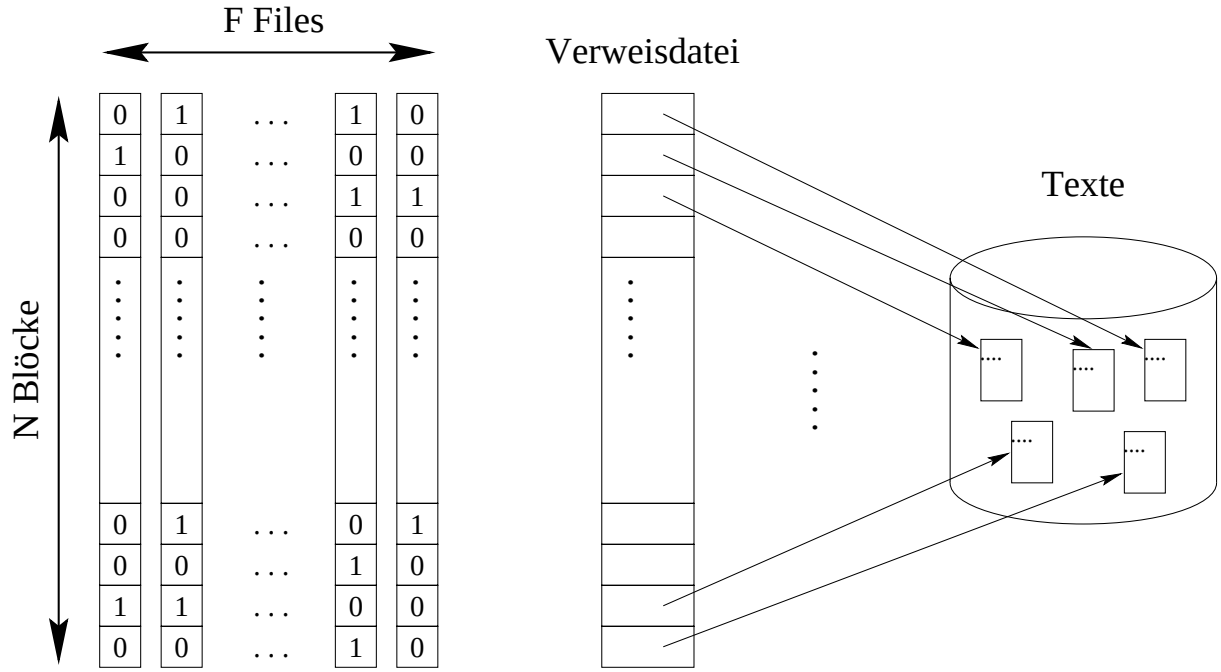
Veranschaulichung eines SSF:



Bitscheibenorganisation:

- Beim SSF werden die Blocksignaturen stets komplett gelesen.
- Für den Abgleich zwischen Anfragesignatur und Blocksignatur werden aber nicht alle Bits der Blocksignatur benötigt, sondern nur die Bits der Positionen, die in der Anfragesignatur gesetzt sind.
- Bei der Bitscheibenorganisation (Bit-Sliced Signature Files, BSSF) findet eine vertikale Partitionierung des Signature-Files statt,
  d.h. die einzelnen Bits einer Signatur werden in F verschiedenen Dateien abgelegt.
- Jede Datei entspricht einer bestimmten Bitposition in einer Signatur (Bitscheibe) bzw. einer Spalte der $N \times F$-Matrix.
- Bei einer einfachen Anfrage braucht man nun nur die Einträge der m Bitscheiben zu lesen, deren Bits in der Anfragesignatur gesetzt sind.
- Durch eine horizontale AND-Verknüpfung der Einträge dieser Bitscheiben erhält man die Kandidatenmenge.

Veranschaulichung eines BSSF:



Beispiel: Retrieval mit Hilfe eines BSSF

Es gelten die Wortsignaturen:

text	010 010 001 000
search	010 000 100 100
method	100 100 000 001
full	010 110 000 000
knowledge	000 111 000 000
retrieval	000 000 111 000
lexicon	100 010 010 000

Anfrage: text search 010 010 101 100

Es müssen also nur die 2-te, 5-te, 7-te, 9-te und 10-te Bitscheibe gelesen werden.

Bitscheibe Nr. →	2	5	7	9	10	
text search method	1	1	1	1	1	
search knowledge retrieval	1	1	1	1	1	← *false drop*
full text search	1	1	1	1	1	
lexicon knowledge retrieval	0	1	1	1	0	← muss nicht untersucht werden

Bemerkungen:

- Das BSSF hat den Nachteil, dass bei Einfügungen F Plattenzugriffe notwendig sind.
- Es ist aber keine Reorganisation notwendig, so dass ein BSSF auch für optische WORM Plattensysteme anwendbar ist.

S-Baum:

- Im Gegensatz zum BSSF findet beim S-Baum eine horizontale Partitionierung der $N \times F$-Matrix statt.
- Der S-Baum soll analog zum B-Baum sowohl effiziente Such- als auch Änderungsoperationen auf einer Menge von Signaturen erlauben.
- Jeder Knoten eines S-Baums besteht aus einer Menge von maximal k Einträgen der Form: *(Signatur, Adresse)*
- Für innere Knoten des S-Baums zeigt *Adresse* auf einen Sohn im S-Baum.
- Für Blätter des S-Baums zeigt *Adresse* auf ein Dokument (oder Block).
- Bei den inneren Knoten besteht die Signatur eines Eintrags aus der Überlagerung (OR) der Signaturen des entsprechenden Sohnes.
- Die Signaturen in den Blättern sind die Signaturen der Dokumente.
- Es sollten stets solche Signaturen in einem Knoten zusammengefasst werden, die eine kleine Hamming-Distanz aufweisen.

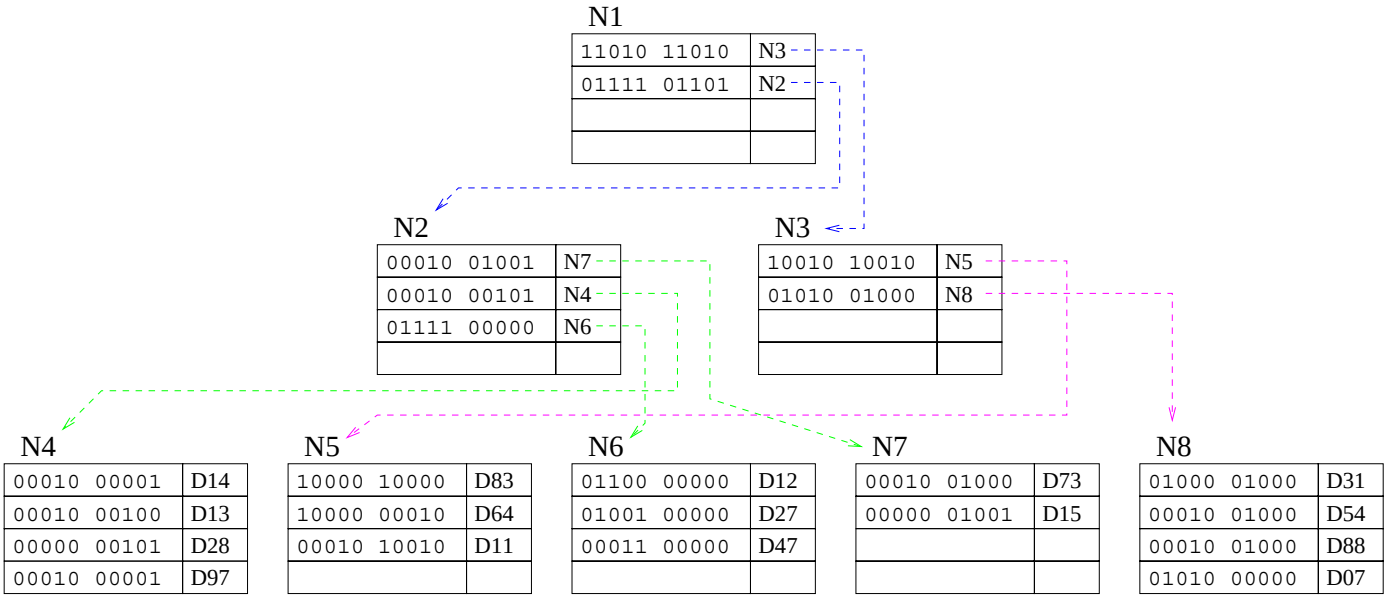
- Die Suche erfolgt ausgehend von der Wurzel.

  Hierbei brauchen jeweils nur die Söhne weiter untersucht zu werden, für die der Signaturabgleich (AND) im Vater erfolgreich war.
- Bei Einfügungen wird ein S-Baum ähnlich wie ein B-Baum ausgeglichen.

  Einfügungen beginnen in den Blättern.
- Hierzu wird ein Knoten ausgewählt, für den im Vater möglichst wenige zusätzliche Bits gesetzt werden müssen.
- Wenn in einem Blattknoten ein Überlauf eintritt, werden die Signaturen auf zwei Knoten verteilt.

  Die Signaturen innerhalb eines Knotens sollten dabei möglichst ähnlich sein.

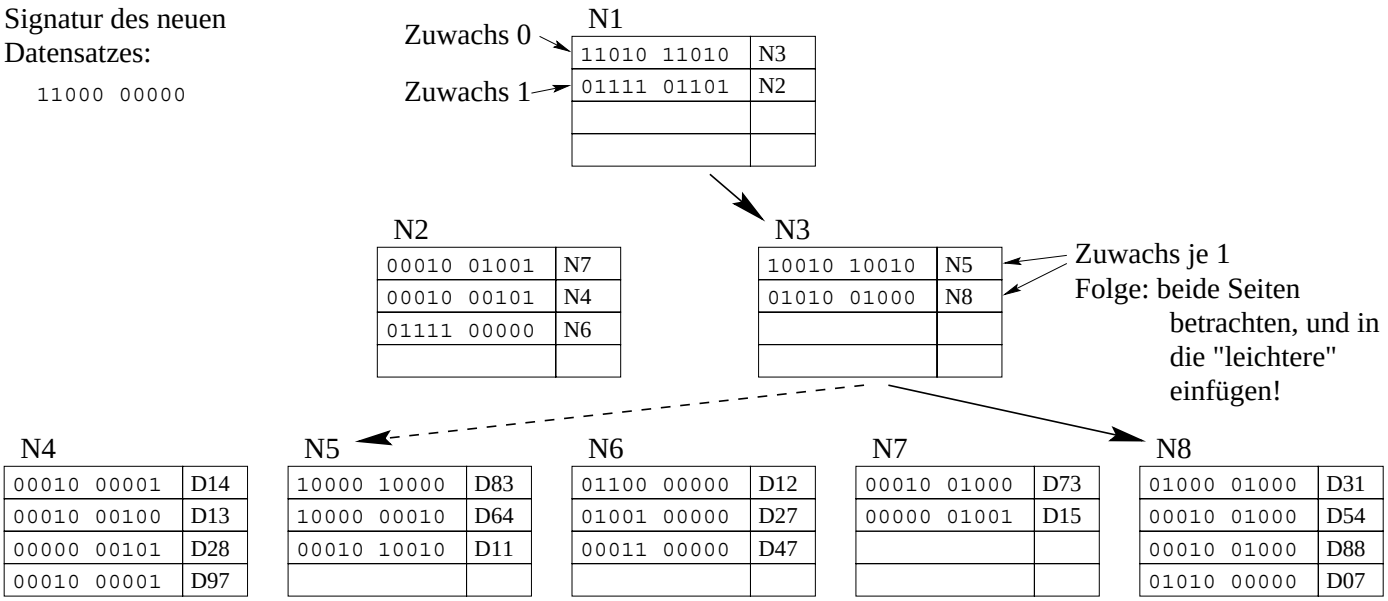
Skizze eines S-Baums:



Beispiel für eine Einfügung:

Signatur des neuen Datensatzes:

11000 00000



Wenn die Seite S_i, in die eingefügt werden muss, bereits voll ist:

- Die Datensätze aus S_i müssen über zwei neue Seiten S_x und S_y verteilt werden.
- Dazu bestimmt man zunächst zwei möglichst verschiedene *Saatsignaturen* und fügt je eine in S_x und S_y ein.

  Z.B. indem man das Paar mit den meisten Unterschieden wählt.

  Bei großen Seiten (mit b Signaturen pro Seite) kann dies wegen des Aufwandes von $O(b^2)$ zum Problem werden. Man kann dann unter allen Signaturen z.B. die wählen, deren gesetzte Bits am weitesten Links (für die eine Saatsignatur) und am weitesten Rechts (für die andere Saatsignatur) liegen.
- Anschließend sucht man unter den restlichen Signaturen in der Seite S_i abwechselnd nach Signaturen, die am besten zu den bisherigen Signaturen in S_x bzw. S_y passen und fügt diese dort ein.
- Der Eintrag in der Vaterseite, der auf S_i gezeigt hat, wird nun durch die Einträge, die auf S_x und S_y zeigen ersetzt. Dabei kann es natürlich erneut zum Überlauf der Vaterseite kommen, der analog durch Aufteilen der Vaterseite bearbeitet wird.
- Gibt es keine Vaterseite, so wird eine neue Wurzel mit zwei Einträgen angelegt.

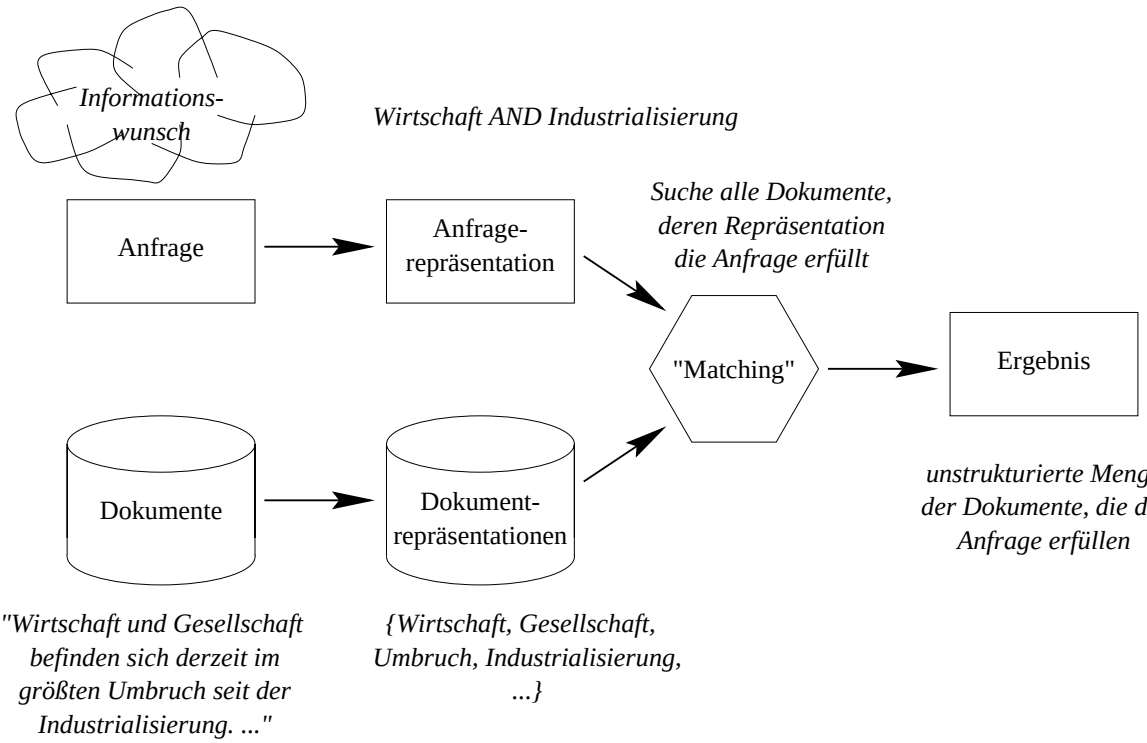
Bemerkungen:

- Vorteil des S-Baum ist, dass eine Suche in sublinearer Zeit möglich ist.
- Nachteile:
  - S-Bäume sind durch die eventuell notwendige Reorganisation (Split) bei Einfügungen nicht für WORM-Systeme anwendbar.
  - Knoten nahe der Wurzel können u.U. viele gesetzte Bits aufweisen und sind somit weitgehend nutzlos.

Vergleich der Speicherungsstrukturen für Signaturen (nach [Fuh96]):

	Profil			
Struktur	Suche	Einfügen	Löschen	Speicher
sequentiell	selten	dominant	selten	wichtig
Bitscheiben	dominant	selten	selten	weniger wichtig
S-Baum	(wenig)	viel	viel	weniger wichtig

6.3 Boolesches Retrieval



Dokumentrepräsentation:

Ein Dokument wird durch die Menge der in ihm auftretenden Wörter repräsentiert.

Dabei sind zahlreiche Varianten denkbar:

- Man kann statt einer Menge eine Multimenge verwenden
- Man kann eine Menge von Paaren (Wort, Vorkommensort) verwalten.
  Der Vorkommensort kann sich dabei beziehen auf
  - Titel, überschrift, normaler Text, ...
  - Wortposition im Text
  - Wortposition im Satz und Satznummer
- Man kann ferner vor der Erstellung der Repräsentation eine Stoppworteliminierung und/oder eine Stammformreduktion durchführen.

Anfragerepräsentation:

Als Anfrage werden zunächst einzelne Wörter akzeptiert. Die Bedeutung ist:

- Suche alle Dokumente, die das Wort enthalten

Ferner können Anfragen mit den üblichen Booleschen Operatoren verknüpft werden.

Beispiel:

Adler AND (Bär OR Löwe)

Implementierungsmöglichkeiten:

- Pattern Matching
- Signaturen (insbesondere für AND-Verknüpfungen)
- Invertierte Listen (als eine der wichtigsten Strukturen)
- ...

6.3.1 Invertierte Listen

Grundidee:

Die **normale Darstellung** ist:

Das Dokument wird mit allen in ihm vorkommenden Wörten gemeinsam abgespeichet.

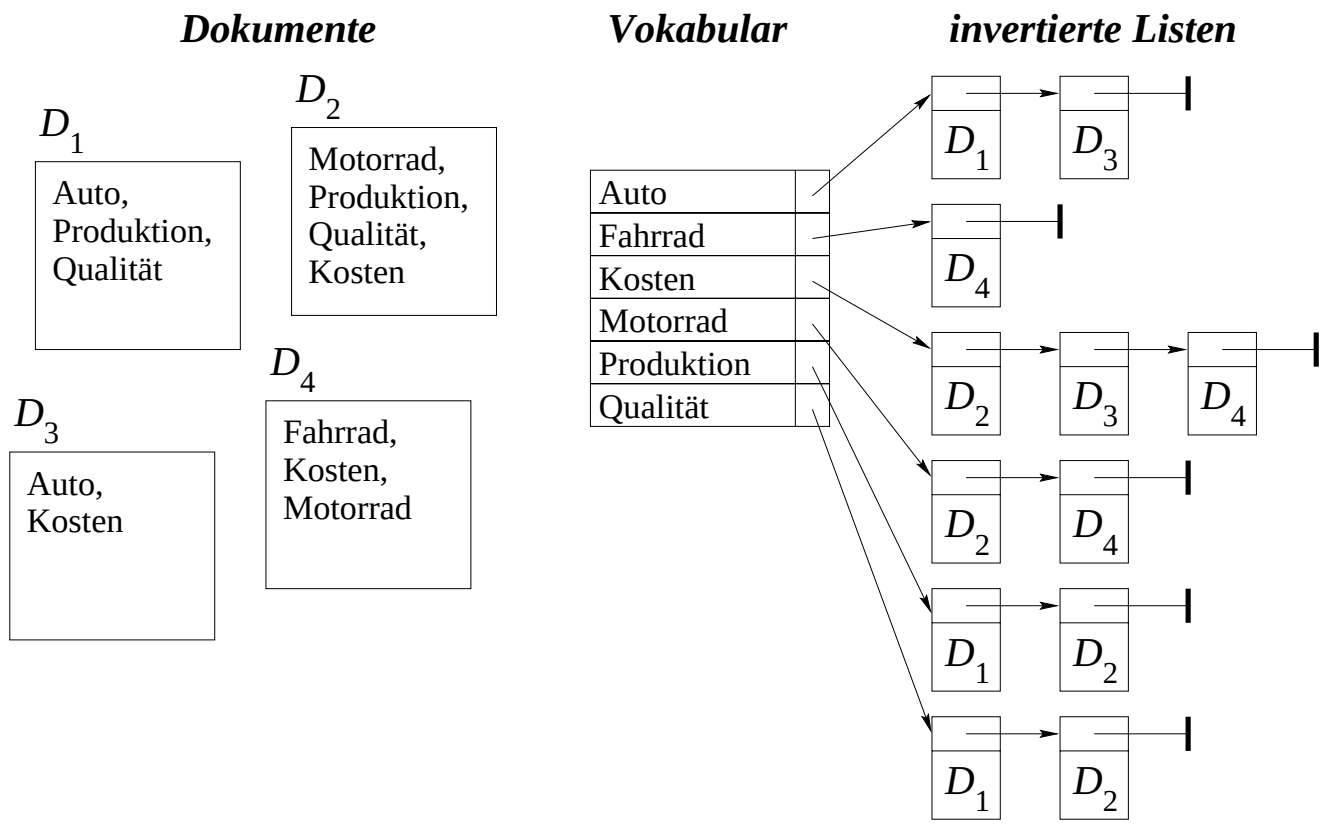
Die Suche geht aber nicht von den Dokumenten, sondern von den Wörten aus!

Also **invertiert** man die Verwaltung:

Man speichert zu jedem Wort alle Dokumente ab,
die dieses Wort enthalten.

Dazu wird zu jedem Wort eine Liste der Dokumente verwaltet, die dieses Wort enthalten:

die invertierte Liste



Die einfachen Operationen des booleschen Information Retrieval lassen sich dann auf einfache Listenoperationen abbilden:

- $q = t_i$
  Ordne q die invertierte Liste von t_i zu.

- $q = q_1\ AND\ q_2$
  Ordne q eine Liste zu, die nur die Elemente enthält, die in den Listen von q_1 und q_2 vorkommen.
  (entspricht der Schnittmenge der beiden Listen)
  Dies kann besonders effizient implementiert werden, wenn man die Listenelemente sortiert verwaltet; z.B. nach Dokumentnummern.
  Dann müssen die Listen von q_1 und q_2 lediglich parallel durchlaufen werden.

- $q = q_1\ OR\ q_2$
  entspricht der Vereinigungsmenge der beiden Listen.

- $q = NOT\ q_2$
  würde bedeuten, eine Liste aufzubauen, die Elemente für alle Dokumente außer den in q_2 enthaltenen beinhaltet,
  da dies i.allg. eine sehr lange Liste ergeben würde, wird oft nur die folgende Operation angeboten:

- $q = q_1\ AND\ NOT\ q_2$
  Dies kann implementiert werden, indem aus der Liste, die q_1 entspricht, die Elemente, die auch in q_2 enthalten sind, entfernt werden.

Andere Operationen wie

- t_i *NEAR[k]* t_j oder
- *IN_TITLE* t_i

können nur mit Hilfe zusätzlicher Informationen in den invertierten Listen verarbeitet werden.

Ggf. erforderliche zusätzliche Informationen:

Vorkommenshäufigkeit im Dokument und/oder Vorkommensort(e) im Dokument

Wieviel Platz benötigt eine invertierte Liste?

Beispiel: TREC Disk 4 Financial Times (564 MB Daten)

- 210 158 Artikel mit im Mittel 412,7 Wörtern
- Pro Eintrag in einer invertierten Liste 4 Byte
- Ergibt: 330,86 MB für die invertierte Liste!

⇒ Hintergrundspeicherstruktur erforderlich!

Zipf's Law:

- Einige Wörter kommen sehr häufig vor:
  - Die beiden häufigsten Wörter können mehr als 10% aller Vorkommen stellen,
  - die sechs häufigsten mehr als 20% und die 50 häufigsten 50 %.
- Sehr viele Wörter sind dagegen sehr selten.
- (Rangzahl in der Häufigkeit) × Vorkommenszahl ist praktisch konstant!

Zipf'sches Gesetz:

Für einen repräsentativen Textkorpus C bezeichne $W(C)$ die Menge der Wörter, die in C vorkommen, und $h(w)$ die Häufigkeit, mit der das Wort $w \in W(C)$ in dem Korpus vorkommt.

$r(w)$ bezeichne den Rangplatz von $w \in W(C)$, wenn die Wörter nach abfallender Häufigkeit sortiert werden.

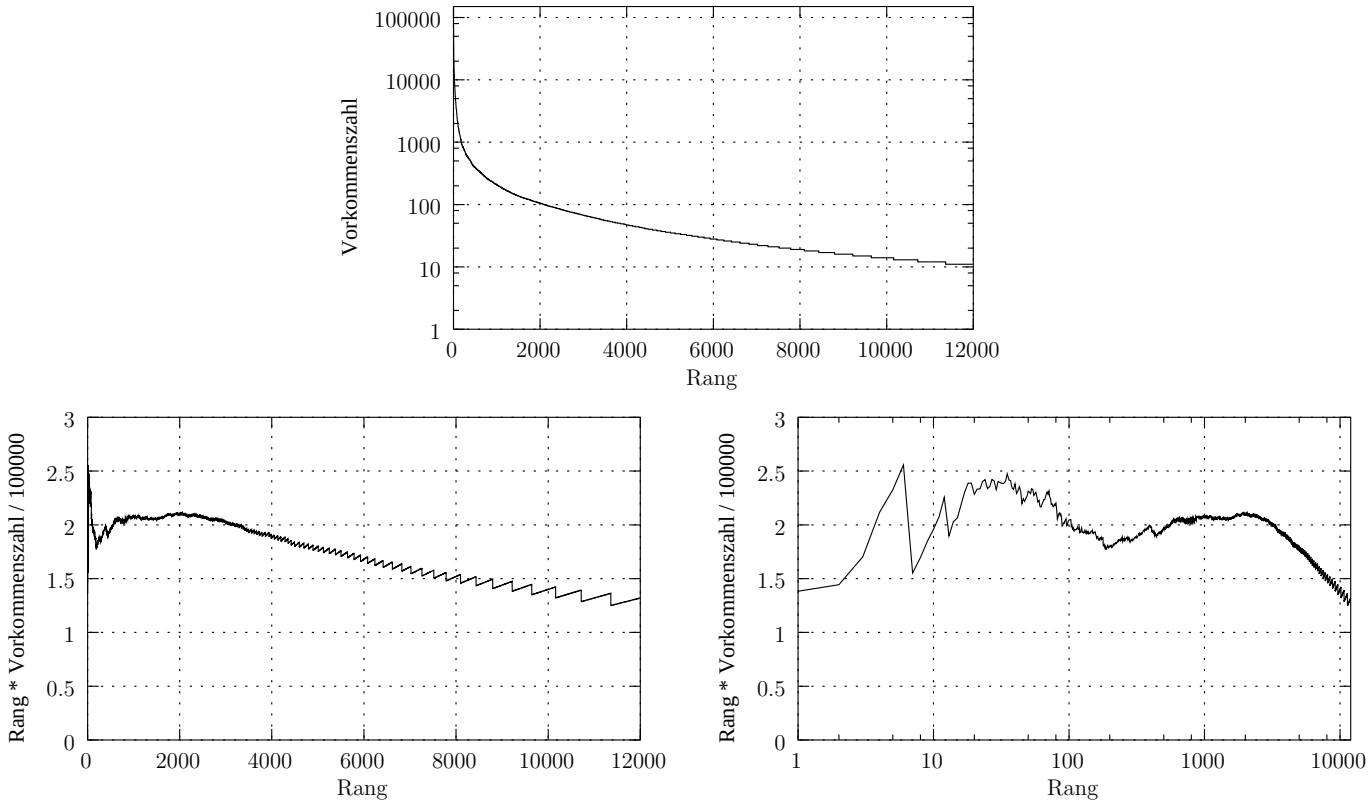
Dann gilt $r(w) \cdot h(w) \approx c = \text{konstant}\ \forall w \in W(C)$

„Zipf's Law" am Beispiel des Brown und des Lob Korpus (nach [Fer99]).

In der dritten Spalte steht das Produkt aus Rang und Häufigkeit (Anzahl) dividiert durch 100000.

- Min: 1.24982
- Max: 2.55618
- Mittel: 1.69661695446236

Rang	Anzahl	$\frac{r(w)\cdot h(w)}{100000}$	Term
1	138323	1.3832	the
2	72159	1.4432	of
3	56750	1.7025	and
4	52941	2.1176	to
5	46523	2.3262	a
6	42603	2.5562	in
7	22177	1.5524	that
⋮	⋮	⋮	⋮
430	443	1.9049	following
431	443	1.9093	short
⋮	⋮	⋮	⋮
2804	73	2.0469	destroy
2805	73	2.0476	determination
⋮	⋮	⋮	⋮
12032	11	1.3235	would-be
12033	11	1.3236	yachting
12034	11	1.3237	yell



Folgen:

- Die invertierten Listen müssen auf dem Sekundärspeicher verwaltet werden.
- Es gibt viele sehr kurze und einige sehr lange Listen.
  Mit anderen Worten, bei einer Seitengröße von z.B. 4KB wird es
  - viele kurze Listen geben, die eine Seite sehr gering auslasten würden, während
  - sich einige lange Listen über mehrere Seiten erstrecken müssen.
- Da das Vokabular aus sehr vielen Begriffen bestehen kann (im Beispiel 12034) muss es auch eine Suchstruktur geben, mit der die Listen gesucht werden können.

Zur Ablage einer kurzen Liste werden dabei ggf. auch Teilbereiche eines Blockes verwendet.

Die einem Term zugeordnete Liste kann adressiert werden durch

- Dateikennung (Name oder Kennnummer)
  (ggf. existiert auch nur eine Datei für alle Listen)
- Blocknummer
- Anfangsadresse des Bereichs für die Liste
- Länge des Bereichs, der für die Liste reserviert ist

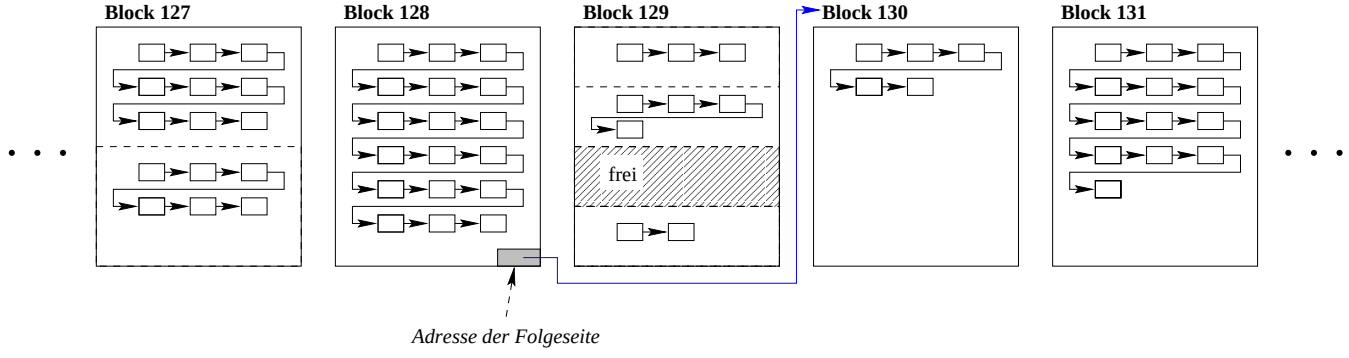
Die Listen für hochfrequente Terme können sich dabei auch über mehrere Blöcke erstrecken. Am Ende der einzelnen Blöcke ist dabei die Nummer des Folgeblocks vermerkt.

Beispiel für die Ablage invertierter Listen in einer Datei:

Verwaltungstabelle

Term	*Blocknum.*	*Länge*	*Startposition*
...	...	...	...
Goldfisch	129	256	0
Hund	128	1024	0
Katze	127	512	512
Kuh	131	1024	0
...	...	...	...
Schlange	129	256	768
Tiger	127	512	0
...	...	...	...
Zebra	127	256	256

Datei mit den Listen



Was ist noch offen?

- *Freispeicherverwaltung* um einer neuen / wachsenden Liste Platz zuweisen zu können.
- *Größe des Bereichs* für eine gegebene Liste
- Realisierung der *Verwaltungstabelle*

Freispeicherverwaltung

- Wir führen für jede Bereichsgröße eine eigene Liste von freien Bereichen.
- Frei werdende Bereiche oder ungenutzte Bereiche, die beim Anlegen eines neuen Blocks entstehen, können so leicht eingefügt werden.
- Wird ein neuer Bereich einer bestimmten Größe benötigt, so wird zunächst die Freiliste für diese Bereichsgröße betrachtet.

  Ist dort kein Eintrag vorhanden, wird ein neuer Block am Ende der Datei angelegt und in Bereiche der benötigten Größe aufgeteilt.

  Einer der Bereiche wird dann sofort verwendet und die anderen in die entsprechende Freiliste eingehängt.

Größe des Bereichs für eine gegebene Liste

Wir gehen davon aus, dass ein initialer Datenbestand existiert, der dann dynamisch verändert wird.

- Nach einer ersten Analyse weiß man wieviele Einträge die initiale Liste zu einem Term haben wird.
- Man reserviert den Bereich (ganze Seite(n), halbe Seite, viertel Seite, ...) dann mit einem gewissen Polster für späteres Wachstum.
- Wird der Bereich trotzdem später durch Berücksichtigung neuer Dokumente zu klein, wird er freigegeben und die Liste in einen neuen größeren Bereich verlegt.

Ist der Bereich für die Liste beim Überlauf bereits ein vollständiger Block, wird ein neuer Folgeblock angehängt.

Bei schrumpfenden Listen lohnt es sich im Allgemeinen nicht, die Liste ggf. in einen kleineren Bereich umzuspeichern, weil man dann bei späterem Wachstum der Liste erneut umspeichern müßte.

Realisierung der Verwaltungstabelle

Die Verwaltungstabelle kann z.B. als B-Baum verwaltet werden.

- Ein B-Baum ist ein sogenannter Vielweg-Suchbaum
- Die typische Aufgabenstellung für einen B-Baum lautet:

Gegeben eine Menge von Datensätzen mit Schlüsseln aus einem geordneten Wertebereich, organisiere diese Menge so, daß ein Datensatz mit gegebenem Schlüssel effizient gefunden, eingefügt oder entfernt werden kann.

Effizient bedeutet hier: mit möglichst wenig Hintergrundspeicherzugriffen.

Datensatz in unserem Fall: <*Term*, Blocknummer, Startposition, Länge>

Der Schlüssel ist der Term selbst.

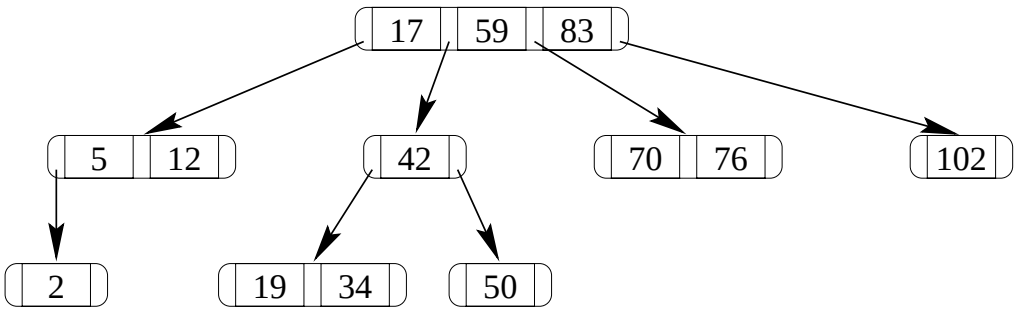
Um die Kosten für eine Suche - die ja der Pfadlänge entsprechen - möglichst gering zu halten, wählt man *Bäume mit einem hohen Verzweigungsgrad.*

Man wählt die Größe der Knoten daher gleich der Größe einer Seite des Hintergrundspeichers.

Man kann leicht zeigen: die minimale Höhe eines Baumes vom Grad d ist $O(\log_d n)$.

Ein *allgemeiner Suchbaum* (auch *Vielweg-Suchbaum* genannt) ist eine Verallgemeinerung des binären Suchbaums.

Beispiel:



Definition: (Vielweg-Suchbäume)

1. Der leere Baum ist ein Vielweg-Suchbaum mit Schlüsselmenge $\emptyset$.
2. Seien $T_0,\ ...,\ T_s$ Vielweg-Suchbäume mit Schlüsselmengen $\overline{T}_0,\ ...,\ \overline{T}_s$ und sei $k_1,\ ...,\ k_s$ eine Folge von Schlüsseln, so dass gilt:

   $$k_1 < k_2 < \ ..., \ < k_s.$$

   Dann ist die Folge

   $$T_0\ k_1\ T_1\ k_2\ T_2\ k_3\ \ ...\ k_s\ T_s$$

   ein Vielweg-Suchbaum genau dann, wenn gilt:

   $$\forall x \in \overline{T}_i:\ k_i < x < k_{i+1} \quad \text{für } i = 1,\ ...,\ s-1$$

   $$\forall x \in \overline{T}_0: \qquad x < k_1$$

   $$\forall x \in \overline{T}_s: \qquad k_s < x$$

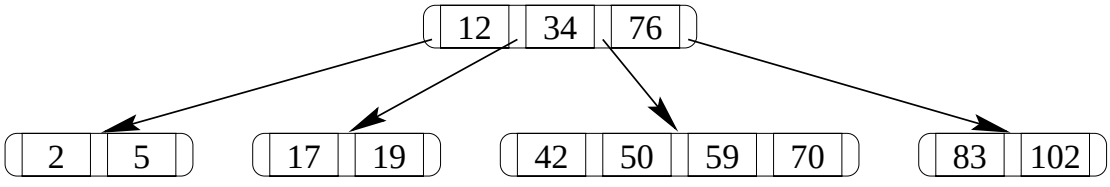
   Seine Schlüsselmenge ist $\{k_1,\ ...,\ k_s\} \cup \cup_{i=0}^{s} \overline{T}_i$

Definition: (B-Bäume)

Ein B-Baum der Ordnung m ist ein Vielweg-Suchbaum mit folgenden Eigenschaften:

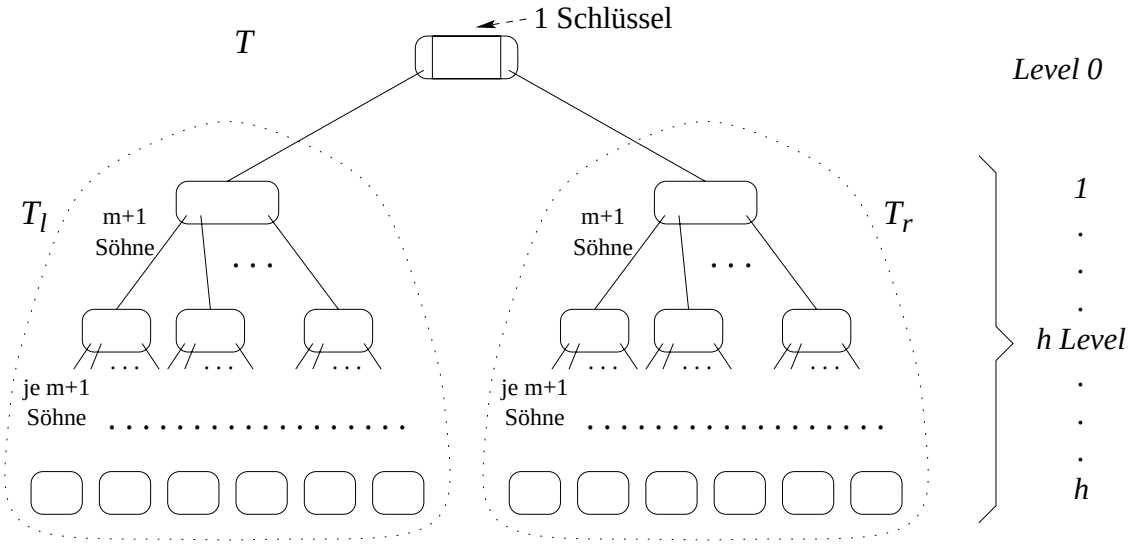
1. Die Anzahl der Schlüssel in jedem Knoten außer der Wurzel liegt zwischen m und $2m$. Die Wurzel enthält ebenfalls maximal $2m$ Schlüssel; wenn sie kein Blatt ist, enthält sie mindestens einen Schlüssel.
2. Alle Pfadlängen von der Wurzel zu einem Blatt sind gleich.
3. Jeder innere Knoten mit s Schlüsseln hat genau $s+1$ Söhne (d.h. es gibt keine leeren Teilbäume).

Für $m = 2$ würde ein B-Baum für die Schlüsselmenge aus unserem Beispiel wie folgt aussehen:



Aus der Strukturdefinition können wir eine obere Schranke für die Höhe eines B-Baumes der Ordnung m mit n Schlüsseln ableiten.

Wir betrachten dazu einen einmal gefüllten B-Baum der Höhe $h+1$:



T_l und T_r sind jeweils vollständige Bäume vom minimalen Grad $(m+1)$.
Das Symbol „#“ stehe für „Anzahl“:

$$\begin{aligned}
\#\text{ Knoten}(T_l) &= 1 + (m+1) + (m+1)^2 + \ldots + (m+1)^{h-1} \\
&= \tfrac{(m+1)^h}{(m+1)-1} = \tfrac{(m+1)^h-1}{m} \\
\#\text{ Schlüssel}(T_l) &= m \cdot \tfrac{(m+1)^h-1}{m} \\
&= (m+1)^h - 1 \\
\#\text{ Schlüssel}(T) &= 2 \cdot (m+1)^h - 1
\end{aligned}$$

Seien nun n Schlüssel in einem Baum der Höhe $h+1$ gespeichert. Es gilt:

$$\begin{aligned}
2 \cdot (m+1)^h - 1 &\leq n \\
(m+1)^h &\leq \tfrac{n+1}{2} \\
h &\leq \log_{(m+1)}\left(\tfrac{n+1}{2}\right) \\
&\in O\left(\log_{(m+1)}(n)\right)
\end{aligned}$$

Mögliche programmiersprachliche Struktur eines Knotens (einer Speicherseite):

type $BNode$ = **record** $used$: $1..2m$;
$keys$: **array**$[1..2m]$ **of** $keytype$;
$infos$:**array**$[1..2m]$ **of** $infotype$;
$sons$: **array**$[0..2m]$ **of** $BNodeAddress$;
end

Das Feld *used* gibt an, wieviele Schlüssel im Knoten gespeichert sind.

Die Zeiger auf Söhne vom Typ $BNodeAddress$ sind logische Seitennummern, die z.B. mit der Formel $BNodeAddress \times$ Blockgröße in die Anfangsposition des Blocks in der Datei umgerechnet werden können.

Für *keytype* und *infotype* ist angenommen, daß Werte fester Länge gespeichert werden.

Um zu gewährleisten, dass ein Knoten in einen Block auf dem Hintergrundspeicher passt, muss man dann m so bestimmen, dass gilt:

$$\begin{aligned} \text{Blockgröße} \geq\ & sizeof(info) \\ & +2m \times sizeof(keytype) \\ & +2m \times sizeof(infotype) \\ & +(2m+1) \times sizeof(BNodeAddress) \end{aligned}$$

Sei in unserem Fall

- die Blockgröße, die das Dateisystem verwendet 8192 Byte,
- der Speicherplatz für die Variable *used* 4 Byte,
- die Länge des Bereichs für einen Term (*keytype*) fix mit 20 Byte angenommen,
- die Länge des Bereichs für die Info aus Blocknummer, Anfangsadresse und Länge (*infotype*) mit 12 Byte angenommen, und
- die Länge einer $BNodeAddress$ mit 4 Byte angenommen,

ergibt sich:

$$\begin{aligned} 8192 &\geq 4 \\ &\quad +2m \times 20 \\ &\quad +2m \times 12 \\ &\quad +(2m+1) \times 4 \\ 8192 &\geq 8 + 2m \times 36 \\ 113,66\ldots &\geq m \end{aligned}$$

Und damit ein Wert von $m = 113$

Haben die Schlüssel oder die Infos keine fixe Länge, kann man die Knoten nicht in dieser Weise als RECORD auffassen.

Sie stellen dann eine Folge von k Bytes dar, in denen hintereinander Sätze variabler Länge stehen, die jeweils mit einem Feld, in dem ihre Länge steht, beginnen.

Die Zahl der Sätze, die in eine Seite passen, ist dann natürlich nicht mehr fix.

Suchalgorithmus

Der folgende Algorithmus sucht nach dem Eintrag mit Schlüssel x:

1. Setze *node* zur Wurzel des B-Baumes.
2. Falls *node* = „NIL“: Der Eintrag ist nicht im Baum; STOP!
3. Setze i zu 1.
4. Solange $i \leq node.used$ und $node.keys[i] < x$ erhöhe i um 1.
5. Falls $i > node.used$: Setze *node* zu $node.sons[node.used]$; fahre mit 2. fort.
6. Falls $node.keys[i] = x$: Der Eintrag ist gefunden; STOP!
7. Falls $node.keys[i] > x$: Setze *node* zu $node.sons[i-1]$; fahre mit 2. fort.

Dabei kann, sofern es sich bei den abgelegten Sätzen um Sätze fester Länge handelt, innerhalb der Knoten natürlich auch binär gesucht werden.

Die Kosten für die Suche sind offensichtlich durch die Höhe des Baumes beschränkt.

Die Eleganz des B-Baumes liegt darin, dass die Struktur auch unter Änderungsoperationen (Einfügen, Löschen) recht einfach erhalten werden kann.

Die Algorithmen für das Einfügen und Entfernen verletzen temporär die Struktur des B-Baumes, indem

- Knoten mit $2m + 1$ Schlüsseln entstehen (diese Verletzung heißt *Overflow*) oder
- Knoten mit $m - 1$ Schlüsseln entstehen (*Underflow*).

Dann wird eine Overflow- oder Underflow-Behandlung eingeleitet, die jeweils die Struktur wieder in Ordnung bringt.

Overflow

Ein Overflow eines Knotens p wird mit einer Operation *split*(p) behandelt.

split(p) teilt den Knoten p mit $2m + 1$ Schlüsseln am mittleren Schlüssel k_{m+1}, so dass Knoten mit Schlüsselfolgen

- k_1 ... k_m und
- k_{m+2} ... k_{2m+1}

entstehen, die jeweils m Schlüssel enthalten.

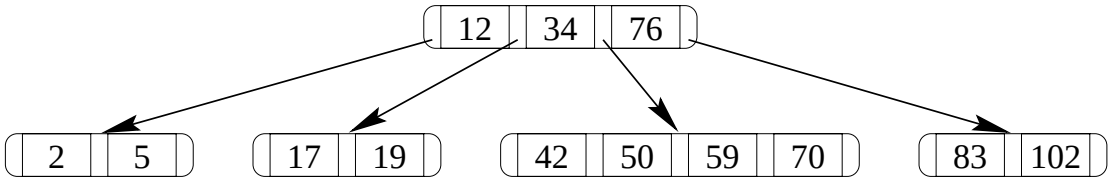
k_{m+1} wandert nach „oben“, entweder

- in den Vaterknoten oder
- in einen neuen Wurzelknoten.

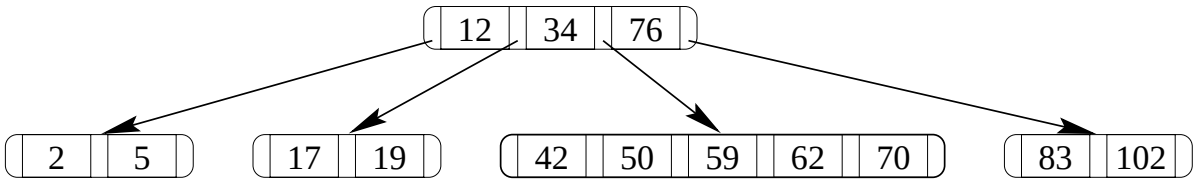
Dadurch kann ggf. der Vaterknoten überlaufen, was wieder einem Overflow entspricht, der in gleicher Weise behandelt wird.

Die Behandlung eines Overflow kann sich also von einem Blatt bis zur Wurzel des Baumes fortpflanzen.

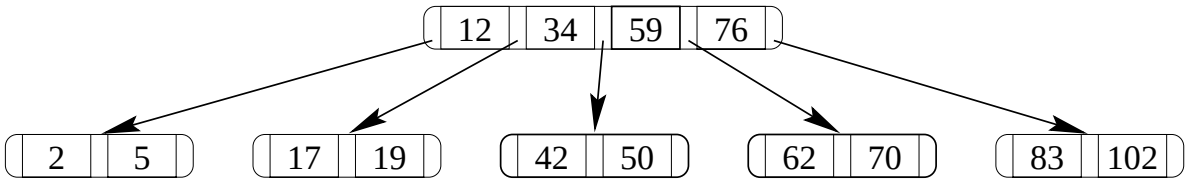
Beispiel für einen Overflow:



Einfügen von 62:



Aufspalten des übergelaufenen Knotens:



Underflow

Ein *Nachbar* eines Knotens sei ein direkt benachbarter Bruder.

Um einen Underflow in p zu behandeln, werden der oder die Nachbarn von p betrachtet.

- Wenn einer der Nachbarn genügend Schlüssel hat, wird seine Schlüsselfolge mit der von p ausgeglichen, so dass beide etwa gleichviele Schlüssel haben.

  Dazu werden
  - die Schlüssel entsprechend neu verteilt und
  - insbesondere der neue „mittlere“ Schlüssel mit dem bisherigen „Trennschlüssel“ im Vater der beiden Knoten ausgetauscht.
- Anderenfalls wird p mit dem Nachbarn zu einem einzigen Knoten verschmolzen, wodurch dann auch der „Trennschlüssel“ im Vater gelöscht werden muss.

Auch die Löschoperation kann sich also bis zur Wurzel fortpflanzen, wenn jeweils eine Verschmelzung durchgeführt werden muß.

Die Kosten für eine Einfüge- oder Löschoperation sind offensichtlich proportional zur Höhe des Baumes.

Der B-Baum unterstützt also Suchen, Einfügen und Löschen in $O(\log_{(m+1)} n)$ Zeit, der Platzbedarf ist $O(n)$.

Die Speicherplatzauslastung ist garantiert besser als 50% (abgesehen von der Wurzel).

Die durchschnittliche Speicherplatzauslastung liegt unter Gleichverteilungsannahmen bei $\ln 2 \approx 69,3\%$.

Beim praktischen Einsatz von B-Bäumen werden oft Varianten des gezeigten Grundschemas verwendet.

B^+-Baum

Beim B^+-Baum werden in den inneren Knoten keine Datensätze gespeichert, sondern nur Trennschlüssel.

Diese Trennschlüssel werden i.a. tatsächlich vorkommende Schlüsselwerte sein, wobei die Absprache ist, daß man bei Gleichheit des gesuchten Schlüsselwertes mit dem Trennschlüssel nach links verzweigt.

Der Baum besteht dann aus

- inneren Knoten, in denen nur Verweise und Trennschlüssel stehen und
- Blättern, in denen die eigentlichen Daten stehen.

Der Vorteil ist, daß so die Einträge in den inneren Knoten klein gehalten und damit die mögliche Verzweigung gesteigert werden kann.

Ein B^+-Baum ist dadurch i.a. bei gleicher Anzahl an Einträgen flacher als ein B-Baum.

B-Baum*

Beim B*-Baum wird vor einem Split eines Knotens zunächst untersucht, ob durch

- Verschiebung von Datensätzen in einen Nachbarn und gleichzeitige
- Anpassung des Trennschlüssels

der Split verhindert werden kann.

Dadurch wird die Speicherplatzauslastung erhöht.

Wir können nun also einen B-Baum verwenden, um die Verwaltungstabelle, die uns von einem gegebenen Term zu der entsprechenden invertierten Liste führt effizient zu implementieren.

Schwächen des booleschen Retrieval:

- keine Rückführung der Wörter auf eine Grundform
- keine Gewichtung der Wörter
  - nach dem Ort des Vorkommens
  - nach der Häufigkeit des Vorkommens
- keine Zerlegung von Mehrwortgruppen
- relativ aufwendige Formulierung der Anfrage
- kaum vorhersehbare Ergebnisgröße
- kein Ranking der Dokumente

Können nur zum Teil abgemildert werden. Insbesondere das fehlende Renking schmerzt!

6.4 Coordination-Level-Match

Hauptproblem des Booleschen IR: Die Ergebnismenge ist unstrukturiert!

Beispiel:

Anfrage: Haus $\wedge$ Garten $\wedge$ Italien

Dokumente, die zwei der gesuchten Begriffe enthalten, werden genauso verworfen, wie Dokumente, die keinen der gesuchten Begriffe enthalten.

Beim Coordination Level Match besteht die Anfrage aus

- Termen, die in den Dokumenten vorkommen sollen und
- Termen, die nicht in den Dokumenten vorkommen sollen

Die Retrievalfunktion berechnet dann zu jedem Dokument eine Kennzahl:

- Begriffe, die in der Anfrage gewünscht werden und die auch im Dokument vorkommen, werden dabei mit einem positiven Wert (z.B. 1) berücksichtigt.
- für Begriffe, die laut Anfrage nicht im Dokument enthalten sein sollen, wird ein negativer Wert (z.B. -1) berücksichtigt.

Die Werte werden addiert.

Beispiel:

- zur Anfrage
  Haus, Garten, NOT Frankreich
  würde sich für das Dokument
  „Garten in Frankreich“
  ein Wert von 0 ergeben.

Implementierung:

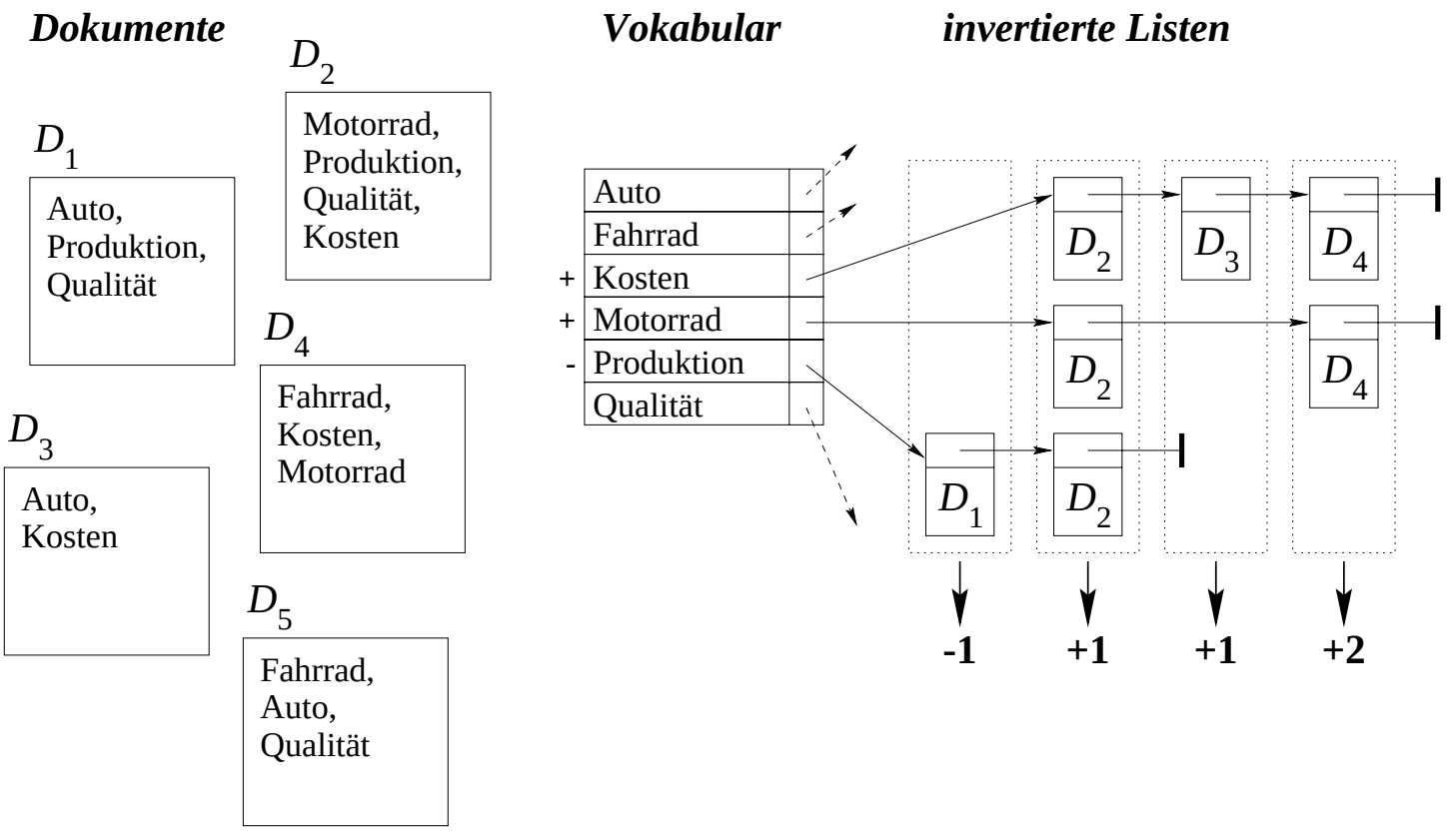
- Man kann die Implementierung recht einfach über invertierte Listen vornehmen.
- Vorteilhaft ist dabei, wenn die Listeneinträge in allen Listen nach Dokumentnummern sortiert sind.
- Man kann dann die Listen zu den in der Anfrage vorkommenden Termen parallel durchlaufen und zu den Dokumenten nacheinander die Kennzahlen berechnen.

Beispiel:

Anfrage: *Kosten, Motorrad, NOT Produktion*

Dokumente, die in keiner der Listen vorkommen werden ebenso wie Dokumente, die einen Wert ≤ 0 erzielen nicht in das Ergebnis aufgenommen.

Die anderen Dokumente werden absteigend nach Werten sortiert.



6.5 Fuzzy-Set Modell

Insbesondere beim Booleschen Retrieval wird davon ausgegangen, dass die Zugehörigkeit eines Elements zu einer Menge immer eindeutig entscheidbar ist.

So wird einem Term t_i letztlich eine Menge von Dokumenten zugeordnet, die er repräsentiert.

Diese Menge entspricht den Dokumenten, die in der invertierten Liste zu t_i enthalten sind, und damit der Menge der Dokumente, in denen t_i vorkommt.

Dies trägt aber in keiner Weise der Unschärfe in der Begriffsverwendung Rechnung.

So kann ein Begriff z.B. auch dann ein guter Repräsentant für ein Dokument sein, wenn er nicht in diesem Dokument vorkommt.

Letztlich erscheint es sinnvoller jedem Dokument einen Grad der Zugehörigkeit zu der Menge $\mathcal{T}_i$ der Dokumente, die durch t_i repräsentiert werden, zuzuordnen, der zwischen 0 und 1 liegen kann.

D.h. wir geben für jedes Dokument D_j und jeden Term t_i aus dem Vokabular einen Wert $\mu_{i,j} \in [0;1]$ an, der angibt, wie repräsentativ t_i für D_j ist.

Damit betrachten wir die Menge der Dokumente, die durch einen Term t_i repräsentiert werden, als Fuzzy-Set.

Definition: Fuzzy-Set

Ein Fuzzy-Set S über einer Grundgesamtheit G wird durch eine Funktion $\mu_S : G \rightarrow [0;1]$ beschrieben, die für jedes Element aus G den Grad der Zugehörigkeit zu S angibt.

Basierend auf dieser Definition werden die Negation, die ODER-Verknüpfung und die UND-Verknüpfung auf Fuzzy-Sets dann üblicherweise wie folgt definiert:

$$\begin{aligned} \mu_{\overline{S}}(u) &= 1 - \mu_S(u) \\ \mu_{A \cup B}(u) &= \max(\mu_A(u), \mu_B(u)) \\ \mu_{A \cap B}(u) &= \min(\mu_A(u), \mu_B(u)) \end{aligned}$$

Anfragen können jetzt weiter im Stile des Booleschen Retrieval gestellt werden.

Ein Dokument D_j wird durch die Werte $\mu_{i,j}$ für alle Terme des Vokabulars repräsentiert.

Beispiel:

Das Vokabular bestehe aus vier Termen und wir betrachten fünf Dokumente.

In der folgenden Tabelle sind die $\mu_{i,j}$ für diese Konstellation angegeben (die – noch – vom Himmel fallen).

Anfrage: Haus $\wedge$ (Italien $\vee$ Frankreich) $\wedge \neg$ Garten

Term	D_1	D_2	D_3	D_4	D_5
Haus	0,5	0,0	0,7	0,3	0,9
Garten	0,9	1,0	0,1	0,0	0,0
Italien	1,0	0,0	0,0	1,0	0,0
Frankreich	1,0	1,0	0,4	1,0	0,5
Anfrage	0,1	0,0	0,4	0,3	0,5

Wie können nun die $\mu_{i,j}$ bestimmt werden (nach [OMK91])

Wir bauen dazu eine Term×Term-Matrix auf.

In diese Matrix tragen wir die Korrelationen zwischen den einzelnen Termen ein.

Sei n_i die Anzahl der Dokumente, die Term t_i enthalten und n_l die Anzahl der Dokumente, die Term t_l enthalten.

Ferner sei $n_{i,l}$ die Anzahl der Dokumente, die sowohl Term t_i als auch Term t_l enthalten.

Dann berechnen wir die Elemente $c_{i,l}$ der Term×Term-Matrix durch:

$$c_{i,l} = \frac{n_{i,l}}{n_i + n_l - n_{i,l}}$$

Man beachte, dass $c_{i,l} = 1$ für $i = l$ gilt.

Damit kann man den Grad der Zugehörigkeit eines Dokumentes D_j zur Menge $\mathcal{T}_i$ berechnen als:

$$\mu_{i,j} = 1 - \prod_{t_l \in D_j} (1 - c_{i,l})$$

Man kann dies wie folgt interpretieren:

- $c_{i,l}$ ist die Wahrscheinlichkeit der Zugehörigkeit zu $\mathcal{T}_i$ wenn t_l im Dokument vorkommt.
- $1 - c_{i,l}$ ist die Wahrscheinlichkeit der Nicht-Zugehörigkeit zu $\mathcal{T}_i$ wenn t_l im Dokument vorkommt.
- $\prod_{k_l \in D_j} (1 - c_{i,l})$ ist die Wahrscheinlichkeit der Nicht-Zugehörigkeit zu $\mathcal{T}_i$ für D_j.
- $1 - \prod_{k_l \in D_j} (1 - c_{i,l})$ ist die Wahrscheinlichkeit der Zugehörigkeit zu $\mathcal{T}_i$ für D_j.

Implementierung

- Eine Implementierung könnte invertierte Listen nutzen, wobei nun in der Liste zu Term t_i alle Dokumente enthalten sein müssen mit $\mu_{i,j} > 0$.
- Probleme bereitet dabei wieder die Negation.
- Kritisch ist auch, dass insbesondere bei längeren Dokumenten die Listen sehr lang werden, weil es sehr viele $\mu_{i,j} > 0$ gibt.

  Man kann in diesen Fällen einen sinnvollen Grenzwert definieren, bis zu dem die $c_{i,l}$ nur berücksichtigt werden.

Probleme

- Durch die Maximum- und Minimumbildung wird eine extreme Position vertreten.

  Anfrage: Haus $\wedge$ Italien

Term	D_1	D_2
Haus	0,3	0,9
Italien	0,3	0,2
Anfrage	0,3	0,2

  Das System hält also D_1 für relevanter als D_2!

  Es wird keine Substitution berücksichtigt.
- Das Problem der booleschen Anfrageformulierung bleibt.
- Bei der Implementierung ergeben sich gegenüber dem Booleschen Retrieval deutliche Performancenachteile.
- Dynamische Datenbestände bereiten Probleme, weil eigentlich die Term×Term-Matrix jedesmal geändert werden müßte – und damit alle Einträge in den invertierten Listen.

  Man berechnet daher ggf. die Matrix nur für einen repräsentativen Datenbestand.

Kapitel 7

Das Vektorraummodell

Im Gegensatz zum Fuzzy-Retrieval kann beim Vektorraum-Modell auch in der Anfrage eine Gewichtung der Terme vorgenommen werden.

Dazu wird ein t-dimensionaler Raum betrachtet. t ist dabei meist die Anzahl der Terme im Vokabular.

- **Dokument** D repräsentiert durch Vektor: $\mathcal{D} = (w_{d1}, w_{d2}, \ldots, w_{dt})$
- **Anfrage** Q ebenfalls durch einen Vektor repräsentiert: $\mathcal{Q} = (w_{q1}, w_{q2}, \ldots, w_{qt})$
- **Ähnlichkeit** zwischen Anfrage und Dokument z.B. durch das Skalarprodukt: $\text{similarity}(\mathcal{Q}, \mathcal{D}) = \sum_{k=1}^{t} w_{qk} \cdot w_{dk}$

Ausgangspunkt:

- **Vokabular** mit t Begriffen (die, die in den Dokumenten vorkommen)
- Ggf. können dabei nur Wörter nach einer Stammformreduktion betrachtet werden.
- Ggf. kann man hier auch Mehrwortgruppen berücksichtigen.
- Ggf. kann mit Hilfe eines Thesaurus eine Abbildung der im Text vorkommenden Wörter auf Vorzugsbenennungen erfolgen.

  Im Vokabular wären dann nur die Vorzugsbenennungen enthalten.
- ...

Wir brauchen nun einige Definitionen:

N = Anzahl der Dokumente in der Dokumentenkollektion

n_k = Anzahl der Dokumente, die den Begriff/Term k enthalten

tf_{dk} = Vorkommenshäufigkeit von Begriff k in Dokument D

7.1 Das Basismodell

Dokument D repräsentiert durch Vektor: $\mathcal{D} = (w_{d1}, w_{d2}, \ldots, w_{dt})$

Wie soll man die w_{dk} berechnen?

w_{dk} soll angeben, wie gut der Term k das Dokument D beschreibt.

Man nimmt nun an, dass zwei Faktoren hierauf einen Einfluss haben:

1. Die Vorkommenshäufigkeit des Terms im Dokument.

Je höher diese Vorkommenshäufigkeit ist, desto besser beschreibt der Term das Dokument.

Man kann diese Häufigkeit durch tf_{dk} berücksichtigen.

Problem: Die Werte für lange Dokumente sind größer als für kurze.

Beispiel:

- Vokabular: {Haus, Garten, Italien, Frankreich}
- D_1 (recht lang) enthält 50-mal Haus und 5-mal Garten → (50, 5, 0, 0)
- D_2 (recht kurz) enthält 2-mal Garten und 2-mal Italien → (0, 2, 2, 0)
- Anfrage: ungewichtet *Garten und Italien*: (0, 1, 1, 0)
- Das Skalarprodukt ist für D_1 gleich 5 und für D_2 gleich 4, obwohl D_2 sicher einschlägiger ist!

Ausweg: Man normiert die Vektoren $w_{di} = \frac{tf_{di}}{\sqrt{\sum_{j=1}^{t} tf_{dj}^2}}$

2. Die Trennschärfe des Begriffes bezogen auf die Dokumentenkollektion.

Je seltener ein Begriff in der gesamten Kollektion ist, desto wichtiger ist er für die Charakterisierung der Dokumente, in denen er vorkommt.

Man könnte als Kennzahl für die Trennschärfe des Terms t_i nun $\frac{N}{n_i}$ verwenden.

Dies erscheint aber zu extrem:

- Sei $N = 100000$, $n_i = 5$ und $n_j = 50$.
- Dann wäre der Korrekturfaktor für t_i gleich $\frac{N}{n_i} = 20000$ und für t_j gleich $\frac{N}{n_j} = 2000$.
- Dadurch würden bei Verwendung von $w_{dk} = \frac{tf_{dk} \cdot \frac{N}{n_k}}{\sqrt{\sum_{i=1}^{t} \left(tf_{di} \cdot \frac{N}{n_i}\right)^2}}$ die sehr seltenen Terme die Vektoren völlig dominieren.

Ausweg: Man verwendet $\log \frac{N}{n_k}$.

- Dann ist der Korrekturfaktor für t_i gleich $\log 20000 = 4,3$ und für t_j gleich $\log 2000 = 3,3$.

Wir erhalten damit die Formel:

$$w_{dk} = \frac{tf_{dk} \cdot \log \frac{N}{n_k}}{\sqrt{\sum_{i=1}^{t} \left(tf_{di} \cdot \log \frac{N}{n_i}\right)^2}}$$

Diese Formel bezeichnet man auch als *tf·idf*-Formel.

- *tf* steht dabei für *term frequency*
- *idf* steht für *inverse document frequency*

Die **Anfrage** Q wird ebenfalls durch einen Vektor repräsentiert: $\mathcal{Q} = (w_{q1}, w_{q2}, \ldots, w_{qt})$

Wenn die Anfrage als Text gegeben ist (wovon man beim Vektorraummodell ausgeht) könnten wir zur Gewinnung der w_{qk} auf die *tf·idf*-Formel zurückgreifen.

Dies erscheint aber aus mehreren Gründen nicht sinnvoll:

- Sofern ein Term in der Anfrage vorkommt, sollte man diesem immer ein gewisses Grundgewicht beimessen.
- Eine Normierung der Vektoren ist hier nicht erforderlich, weil ja für alle Dokumente der gleiche Anfragevektor verwendet wird.

Salton und Buckley [SB88] schlagen daher die folgende Formel vor:

$$w_{qk} = \begin{cases} \left(0.5 + \frac{0.5 \cdot tf_{qk}}{\max\limits_{1 \le i \le t} tf_{qi}}\right) \cdot \log \frac{N}{n_k} & \text{if} \quad tf_{qk} > 0 \\ 0 & \text{if} \quad tf_{qk} = 0 \end{cases}$$

Die Ähnlichkeit zwischen einer Anfrage und einem Dokument kann dann durch das Skalarprodukt beschrieben werden:

$$\text{similarity}(\mathcal{Q}, \mathcal{D}) = \sum_{k=1}^{t} w_{qk} \cdot w_{dk}$$

Alternativ dazu kann man auch die Normierung der Beschreibungsvektoren für die Dokumente zunächst unterlassen und statt dessen $w'_{dk} = tf_{dk} \cdot \log \frac{N}{n_k}$ verwenden.

Als Ähnlichkeitsmaß kann dann das Cosinusmaß verwendet werden, das den Winkel zwischen dem Dokumentvektor und dem Anfragevektor berechnet.

Dieser Winkel ist ohnehin unabhängig von der Länge der Vektoren.

$$\text{sim}_{cos}(\mathcal{Q}, \mathcal{D}) = \frac{\sum_{k=1}^{t} w_{qk} \cdot w'_{dk}}{\sqrt{\sum_{i=1}^{t} w_{qi}^2} \times \sqrt{\sum_{j=1}^{t} w'^2_{dj}}}$$

Damit wird im Grunde die Normierung von der Erstellung der Vektoren auf die Berechnung der Ähnlichkeit verlegt und auch für den Anfragevektor durchgeführt.

Die relative Bewertung der Dokumente bleibt gleich.

Bei sim_{cos} liegen die Ergebnisse im Intervall $[0; 1]$, wobei hohe Werte eine hohe Ähnlichkeit bedeuten.

Vorteile des Vektorraummodells:

1. einfaches und verständliches Modell
2. einfache Frageformulierung durch natürlichsprachliche Anfrage
3. im Gegensatz zu Modellen des probabilistischen IR unmittelbar auf neue Kollektionen anwendbar
4. empirisch sehr gute Retrievalqualität

Probleme des Vektorraummodells:

- Modell enthält zahlreiche (unrealistische) Annahmen
- probabilistische Fundierung fehlt
- Anpassung an andere Anforderungen (z.B. Dokumente mit Titel und Abstract) nur heuristisch möglich

7.2 Implementierung

- durch invertierte Listen (die gebräuchliche Variante)
- durch mehrdimensionale Zugriffsstrukturen (ein Forschungsansatz)
- durch Signaturen (hat sich als wenig erfolgversprechend erwiesen [SP93])

7.2.1 Basisalgorithmus mit invertierten Listen

Gesucht werden die γ relevantesten Dokumente:

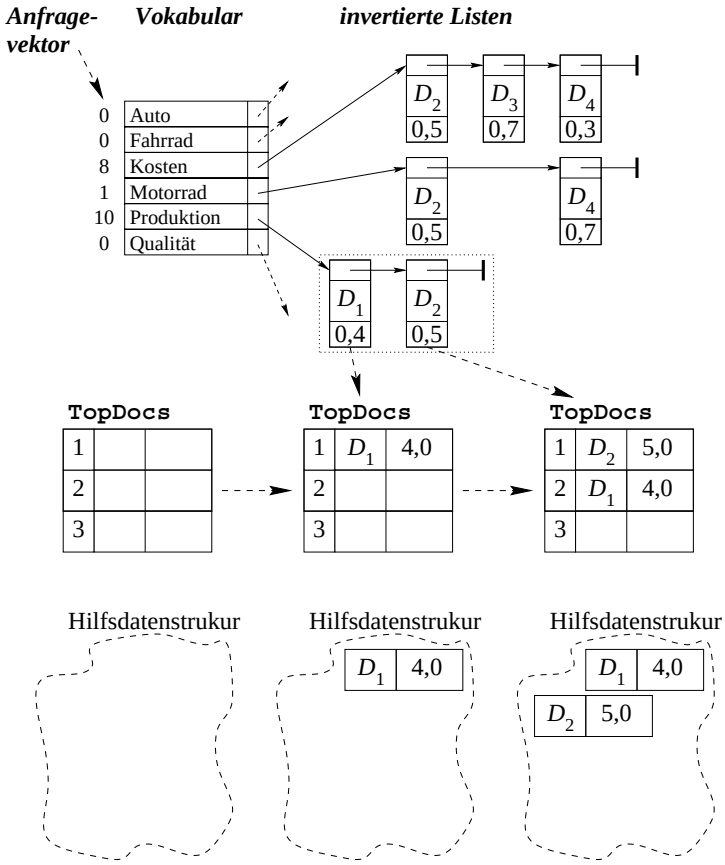
- Betrachte die invertierten Listen zu den Anfragetermen in der Reihenfolge fallender Fragetermgewichte w_{qk}.
- Für jedes Paar (D, w_{dk}) in der gerade betrachteten invertierten Liste addiere $w_{dk} \times w_{qk}$ zum bisherigen Gewicht von D.
  Halte zu den Dokumenten die bisher aufgelaufenen Retrievalgewichte fest.
- Nach dem Ende jeder invertierten Liste prüfe, ob es noch möglich ist, dass das Dokument mit dem aktuellen Rang $\gamma + 1$ Rang γ erreichen kann. NEIN $\Rightarrow$ FERTIG!

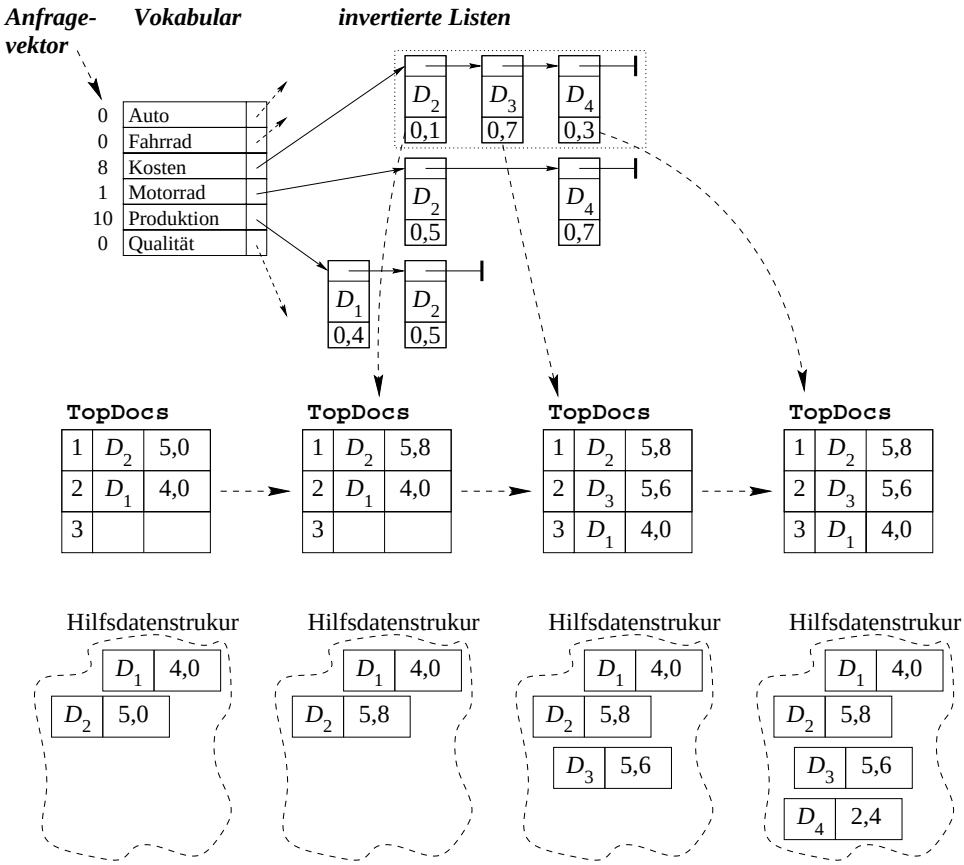
Beispiel:

- Vokabular: { Auto, Fahrrad, Kosten, Motorrad, Produktion, Qualität }
- Anfragevektor: $(0; 0; 8; 1; 10; 0)$
- Die $\gamma + 1$ Dokumente mit den höchsten bisher aufgelaufenen Retrievalgewichten werden in einem Array `TopDocs` nach Gewicht sortiert verwaltet.
- Zu jedem Dokument, für das ein Listeneintrag gelesen wurde, wird das bisher aufgelaufene Retrievalgewicht in einer zusätzlichen Hilfsdatenstruktur verwaltet.

Ablauf des Algorithmus für $\gamma = 2$:

- Zuerst wird die Liste für den Term *Produktion* durchlaufen.
- Anschließend wird die Liste für den Term *Kosten* durchlaufen.
- Danach kann sich an den γ relevantesten Dokumenten nichts mehr ändern, weil der Unterschied zwischen `TopDocs`$[\gamma]$ und `TopDocs`$[\gamma + 1]$ 1,6 ist.
  Da $w_{dk} \leq 1$ gilt und für den Term *Motorrad* nur mit $w_{qk} = 1$ multipliziert wird, brauchen wir daher die Liste für diesen Term gar nicht zu betrachten.





Wir benötigen eine Hilfsdatenstruktur, die Paare (D, g) verwaltet, die bestehen aus

- einem Dokumentdeskriptor D und
- dem bisher aufgelaufenen Retrievalgewicht g.

Auf dieser Struktur seinen folgende Operationen realisiert:

- `IsInDS`(D)
  prüft für einen gegebenen Dokumentdeskriptor D, ob ein Paar zu diesem in der Datenstruktur ist.
- `InsertIntoDS`(D, g)
  Fügt ein neues Paar in die Datenstruktur ein. Zuvor sollte kein Paar für D in der Datenstruktur enthalten gewesen sein.
- `AddToDSEntry`(D, δ)
  Fügt zu dem bisher in der Datenstruktur verwalteten Gewicht für D den Wert δ hinzu.
- `GetWeightFromDS`(D)
  Liefert das derzeit für D in der Datenstruktur abgelegte Gewicht.

Dazu erwarten wir, dass wir auf der invertierten Liste zu einem Term k folgende Operationen ausführen knnen:

- `InitList`(k)
  Initialisiert die Liste für einen sequentiellen Durchlauf und setzt den Durchlaufzeiger auf das erste Listenelement.
- `GetNextElemOfList`(k)
  Liefert das aktuelle Paar (D, w_{dk}) aus der Liste im Rahmen des mit `InitList`(k) gestarteten Durchlaufs und setzt den Durchlaufzeiger um ein Listenelement weiter.
- `NotAtEndOfList`(k)
  Liefert TRUE, sofern der Durchlaufzeiger auf einem Listenelement steht, und FALSE, wenn der Durchlaufzeiger hinter dem letzten Listenelement steht.

Algorithmus zur Suche nach den γ relevantesten Dokumenten [BL85]

for (alle Terme im Anfragevektor mit $w_{qk} > 0$) **do**
  Sei k' der bisher noch nicht betrachtete Term mit dem höchsten Wert $w_{qk'}$;
  `InitList`(k');
  while (`NotAtEndOfList`(k')) **do** /* durchlaufe die Liste */
    (D, $w_{dk'}$) = `GetNextElemOfList`(k');
    if `IsInDS`(D) **then**
      `AddToDSEntry`(D, $w_{qk'} \cdot w_{dk'}$);
    else
      `InsertIntoDS`(D, $w_{qk'} \cdot w_{dk'}$);
    end;
    Sortiere D im Array `TopDocs` gemäß dem Gewicht `GetWeightFromDS`(D) ein;
  end;
  if (`TopDocs`[γ] > `TopDocs`[$\gamma + 1$] + `MaxRemainingWeight`()) **then** FINISH;
end;

Das Array `TopDocs`

- Das Array enthält $\gamma + 1$ Dokumente.
- Es werden zu jedem Zeitpunkt genau die Dokumente verwaltet, die bezogen auf die bisher betrachteten Listeneinträge die höchsten kumulierten Retrievalgewichte haben.
- Bei der Aktualisierung im Algorithmus wird dabei dass betrachtete Dokument an der richtigen Stelle neu einsortiert.

`MaxRemainingWeight`()

- Berechnet den maximalen Gewichtszuwachs, der sich aus der Betrachtung der noch nicht betrachteten Anfrageterme ergeben könnte.
- Wenn wir (aufgrund der Formeln) davon ausgehen, dass die w_{dk} in den Dokumentbeschreibungsvektoren ≤ 1 sind, erhalten wir:

$$\texttt{MaxRemainingWeight}() = \sum_{\text{alle noch nicht betrachteten Anfrageterme } k'} w_{qk'}$$

Anmerkungen zum Algorithmus:

- Der Algorithmus stellt sicher, dass die gefundenen γ Dokumente die γ Dokumente mit den höchsten Retrievalgewichten sind.
- Der Algorithmus stellt *nicht* sicher, dass diese γ Dokumente in der richtigen Reihenfolge im Ergebnis erscheinen!
- Dazu müßte man fordern, dass der Unterschied zwischen allen benachbarten Einträgen in `TopDocs` mindestens `MaxRemainingWeight`() beträgt.
- Dies würde aber die Laufzeit negativ beeinflussen.

Zur Hilfsdatenstruktur, die Paare (D, g) verwaltet:

- Diese Struktur könnte z.B. mit einem AVL-Baum realisiert werden
- Ein Problem kann dabei aber die potentiell sehr große Anzahl der zu verwaltenden Einträge sein.
- Man sollte deshalb auch die Realisierung z.B. als B-Baum erwägen, dessen Laufzeit dann aber durch eine geschickte Pufferstrategie optimiert werden sollte.

7.2.2 Steigerung der Performance durch Verzicht auf Genauigkeit

In der vorgestellten Form arbeitet der Algorithmus, bis die γ Dokumente mit den höchsten Retrievalgewichten sicher erkannt sind.

Da diese Retrievalgewichte aber ohnehin „vage“ sind, kann man eigentlich auch schon früher abbrechen.

Man hat dann zwar noch nicht das rechnerisch richtige Ergebnis erzielt, was aber nichts über die Qualität des Zwischenstandes aussagt.

Man bricht deshalb den beschriebenen Algorithmus mit der Anweisung

if (`TopDocs`$[n]$ > `TopDocs`$[\gamma + 1]$ + `MaxRemainingWeight`()) **then** FINISH;

ab, wobei $n \leq \gamma$ gilt.

Unter den γ Dokumenten, die dann in den ersten γ Zellen von `TopDocs` stehen, sind nur n sicher unter den γ rechnerisch relevantesten.

Der Vorteil ist aber, dass man den Algorithmus früher abbrechen kann:

$$\text{Laufzeitgewinn} \leftrightarrow \text{rechnerische Genauigkeit}$$

Experimentelle Ergebnisse von Buckley & Lewit für die INSPEC-Kollektion

Qualitätsmaß: Recall nach $\gamma = 10$ Dokumenten

n	CPU-Zeit	#gelesene Blöcke	Recall
1	61.3	2833	0.1588
2	68.2	3058	0.1566
3	75.9	3255	0.1555
4	83.1	3451	0.1538
5	90.3	3630	0.1548
6	93.6	3721	0.1543
7	99.7	3863	0.1501
8	104.5	3980	0.1505
9	107.0	4033	0.1508
10	109.2	4111	0.1508

Warum ist für kleine n der Recall sogar besser?

- Der Algorithmus betrachtet zunächst die Anfrageterme mit hohem Anfragegewicht.
- Diese Terme sind in der Regel sehr aussagekräftig.
- Häufig tragen die Anfrageterme mit einem geringeren Gewicht – gerade bei eher langen Anfragen – eher „zur Verwässerung“ der Anfrage bei und senken so die Qualität des Ergebnisses.
- Man kann sich in diesem Zusammenhang aber natürlich auch fragen, ob die Gewichtung der Anfrageterme dann nicht überdacht werden sollte!

7.2.3 Horizontales Durchlaufen der Listen

In [PP97] schlagen Pfeifer und Pennekamp vor, die invertierten Listen zu den Termen mit $w_{qk} > 0$ nicht nacheinander, sondern parallel zu durchlaufen.

Dazu werden die Einträge in den invertierten Listen nicht wie sonst üblich nach den Dokumentidentifikatoren, sondern absteigend nach den w_{dk} sortiert.

Durch die Sortierung ist der w_{dk}-Wert des letzten in der Liste von Term k gelesenen Eintrags eine obere Schranke für die weiteren Einträge in dieser Liste. Diese obere Schranke bezeichnen wir mit m_k.

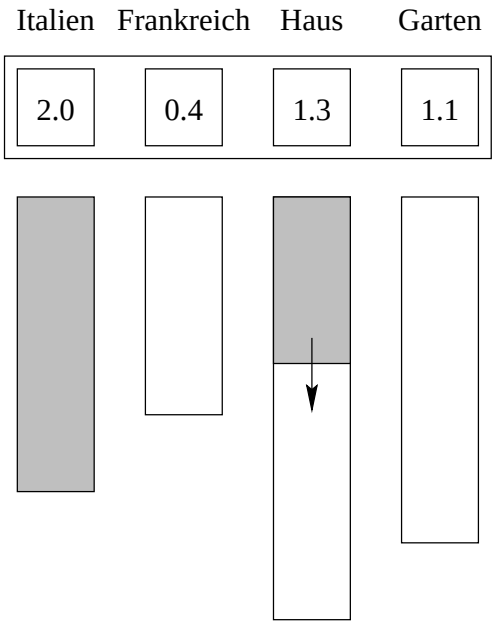
Damit können die noch nicht gelesenen Einträge aus der Liste zu Term k keinen höheren Beitrag zum Retrievalgewicht eines Dokumentes leisten als $w_{qk} \times m_k$.

Beim parallelen Durchlaufen der Listen betrachtet man als nächstes Element immer das Element, der Liste, die aktuell den höchsten Wert für $w_{qk} \times m_k$ aufweist.

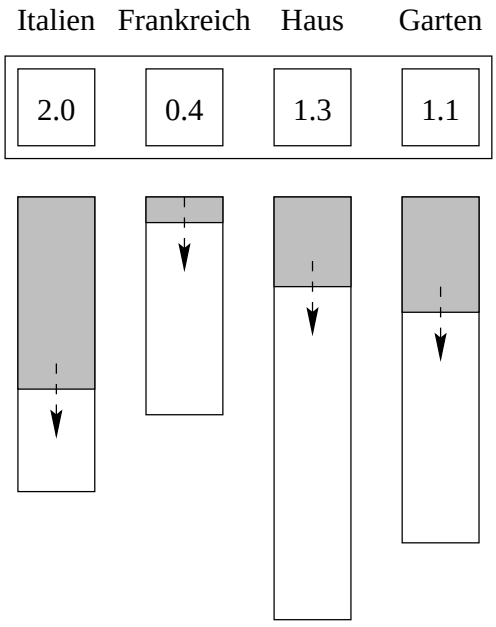
`MaxRemainingWeight`() kann man dabei zu jeder Zeit wie folgt berechnen:

$$\texttt{MaxRemainingWeight}() = \sum_{\{k|w_{qk}>0\}} w_{qk} \times m_k$$

Vertikales Durchlaufen:



Horizontales Durchlaufen:



Vorteile des Verfahrens:

- Man betrachtet als nächstes Element immer dasjenige, das den höchsten Beitrag zum Retrievalgewicht eines Dokumentes leisten kann.
- Dadurch müssen im Allgemeinen weniger Listenelemente betrachtet werden.

Nachteile des Verfahrens:

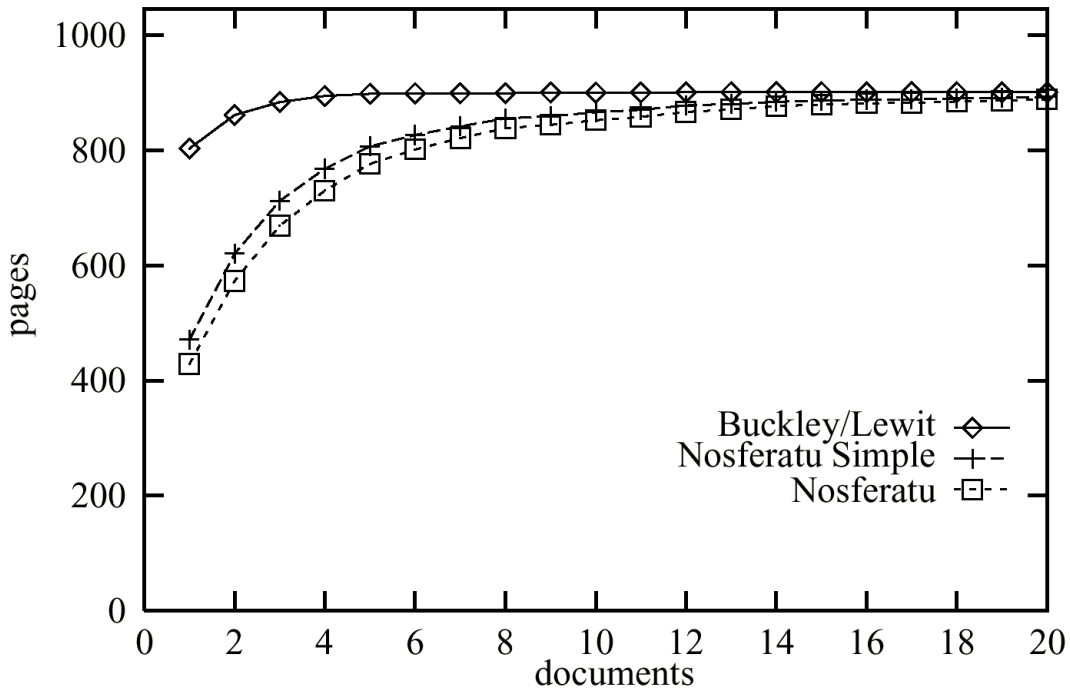
- Während bisher immer nur eine Liste zur Zeit betrachtet werden mußte, müssen nun viele Listen parallel durchlaufen werden.
- Im Extermfall müssen dadurch sogar mehr Blöcke vom Hintergrundspeicher gelesen werden.

Um dem Rechnung zu tragen, schlagen Pfeifer und Pennekamp eine Variante vor, die gelesene Blöcke immer vollständig abarbeitet.

$w_{qk} \times m_k$ wird dabei letztlich verwendet, um die Liste zu bestimmen, für die der nächste Block gelesen werden soll.

Experimente:

- Für die Experimente aus [PP97] wurden die 231201 Dokumente auf der 2. CD von TREC verwendet.
- Die Topics 1 bis 150 wurden dabei als Anfragen mit durchschnittlich 57,4 Termen pro Anfrage verwendet.
- Die Gewichte in den Dokumentbeschreibungsvektoren wurden mit der $tf \cdot idf$-Formel bestimmt.
- Ein Block auf dem Hintergrundspeicher konnte bis zu 1000 Listenelemente aufnehmen.
- Der ursprüngliche Algorithmus von Pfeifer und Pennekamp wird mit *Nosferatu* bezeichnet.
- Die blockweise arbeitende Variante mit *Nosferatu Simple*.



Ergebnis:

- Bei allen Varianten müssen zum Finden der „besten“ 20 Dokumente alle Listen zu Anfragetermen (fast) vollständig durchlaufen werden.
- Für $\gamma < 20$ hat aber die Variante von Pfeifer und Pennekamp Vorteile.
- Je weniger Terme in der Anfrage auftreten, je effizienter sind die Verfahren.

7.2.4 Nutzung einer mehrdimensionalen Zugriffsstruktur

`!!! wird in der Klausur nicht betrachtet !!!`

Motivation

- Es gibt eine Vielzahl effizienter mehrdimensionaler Zugriffsstrukturen für geometrische und geographische Daten (vgl. hierzu [GG98]).
- Zu diesen Strukturen gibt es effiziente Algorithmen zur Bearbeitung von Anfragen, die den nächsten Nachbarn suchen, oder die Datensätze absteigend nach ihrem Abstand zu einem gegebenen Punkt sortieren.
  - für den LSD-Baum [Hen94]
  - für den R-Baum [RKV95]
  - für den Quadtree [HS95]
- Die Zugriffsstrukturen können auch verwendet werden, um zusätzlich Selektionsbedingungen über Standardattributen zu berücksichtigen [HM95].

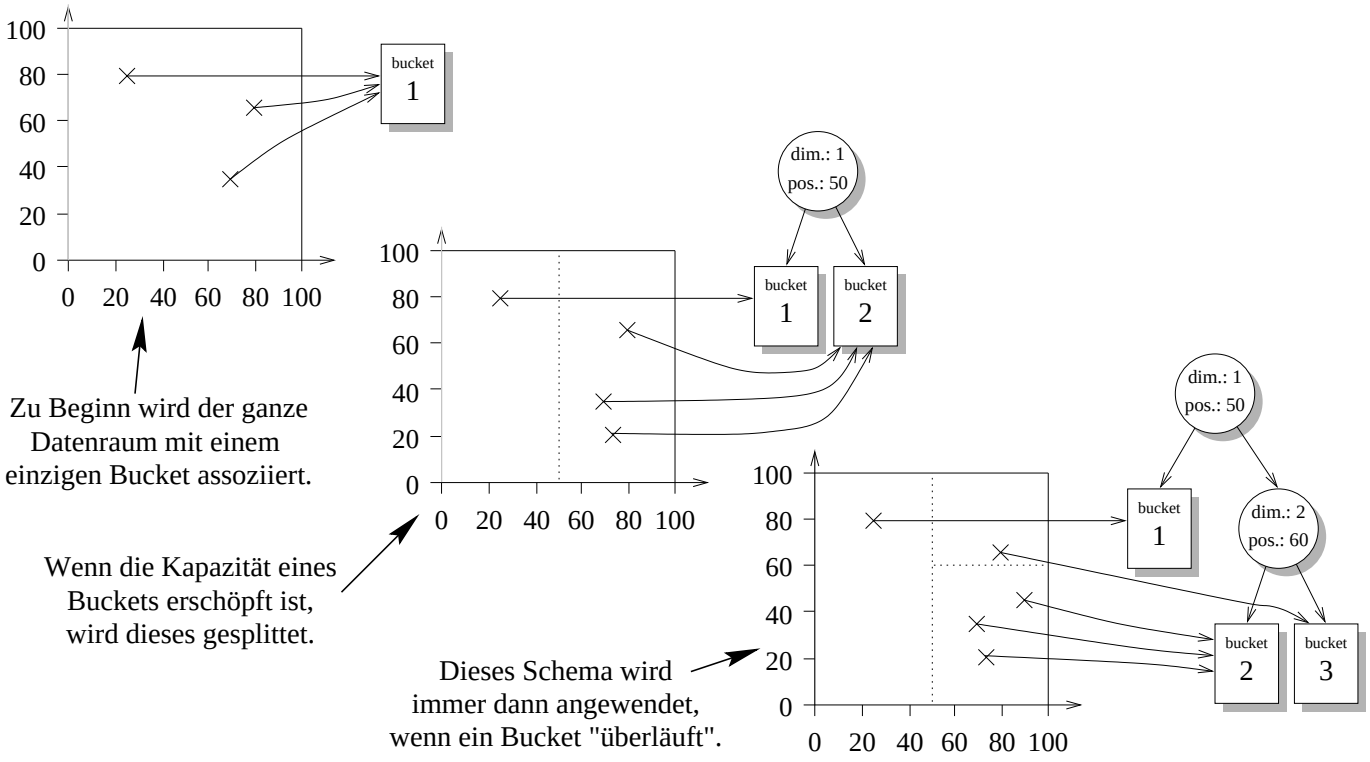
Auf der anderen Seite:

- Im Vektorraummodell werden Dokumente als Punkte in einem hochdimensionalen Raum betrachtet
- Anfragen werden als Ähnlichkeitssuche in diesem Raum bearbeitet.
- Invertierte Listen haben Probleme mit
  - langen Anfragetexten und
  - zusätzlichen Bedingungen über Standardattruibuten.

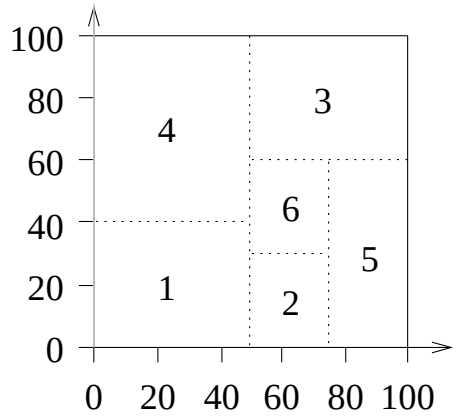
$$\Downarrow$$

Man kann versuchen die Zugriffsstrukturen für geometrische Daten auch beim Vektorraummodell zu nutzen.

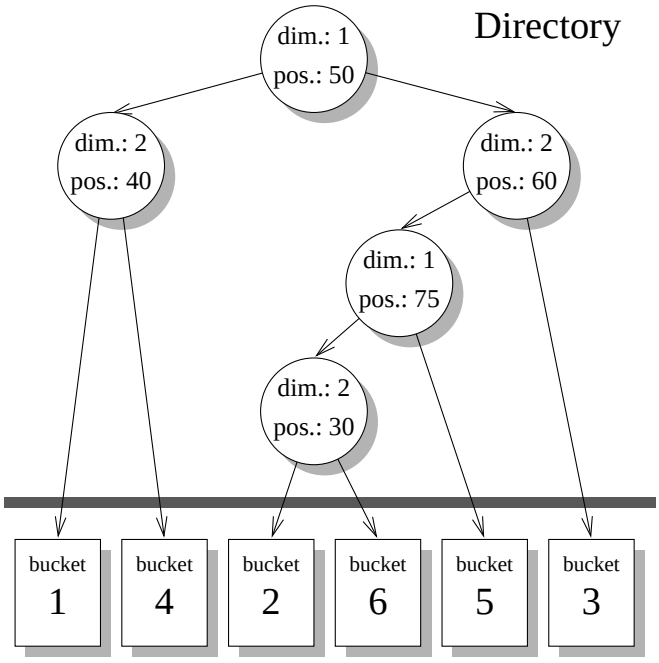
Basiskonzept des LSD-Baumes



Aufteilung des Datenraumes:



Zugehöriger LSD-Baum:

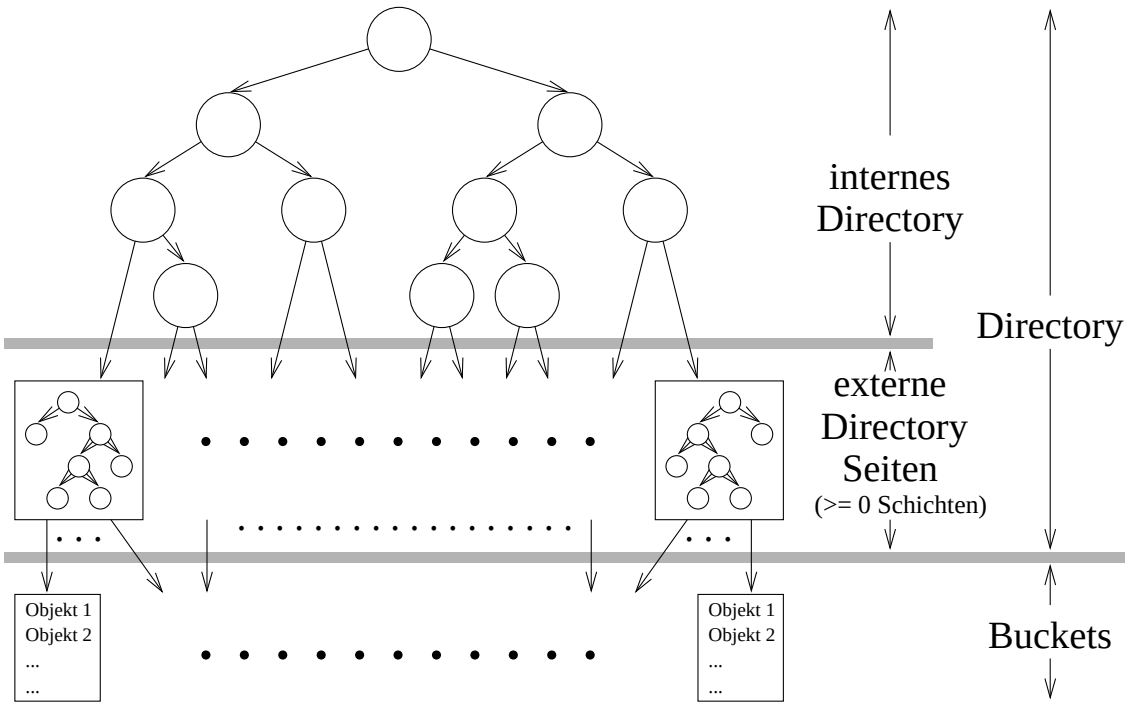


Ansatz zur Verwaltung des Directory auf Hintergrundspeicher [HSW89]:

- Der LSD-Baum verwaltet ein internes Directory mit maximal N_{int} Knoten im Hauptspeicher.
- Wird N_{int} überschritten, wird ein Teilbaum des internen Directory auf eine Seite auf dem Hintergrundspeicher ausgelagert.
- Läuft eine Seite auf dem Hintergrundspeicher über, so werden der rechte und der linke Teilbaum des in dieser Seite gespeicherten Baumes je in einer eigenen Seite gespeichert.

  Die Wurzel wird eine Ebene höher wieder eingefügt.
- Bei der Suche nach einem auszulagernden Teilbaum im internen Directory wird jeweils darauf geachtet, dass eine externe Pfadlängenbalancierung erhalten bleibt.
- Der unterliegende Binärbaum kann aufgrund seiner Multidimensionalität nicht balanciert werden.
- Man muss deshalb durch eine geeignete Wahl der Splitebenen dafür sorgen, dass dieser Baum relativ balanciert bleibt.

Gesamtstruktur eines LSD-Baumes:



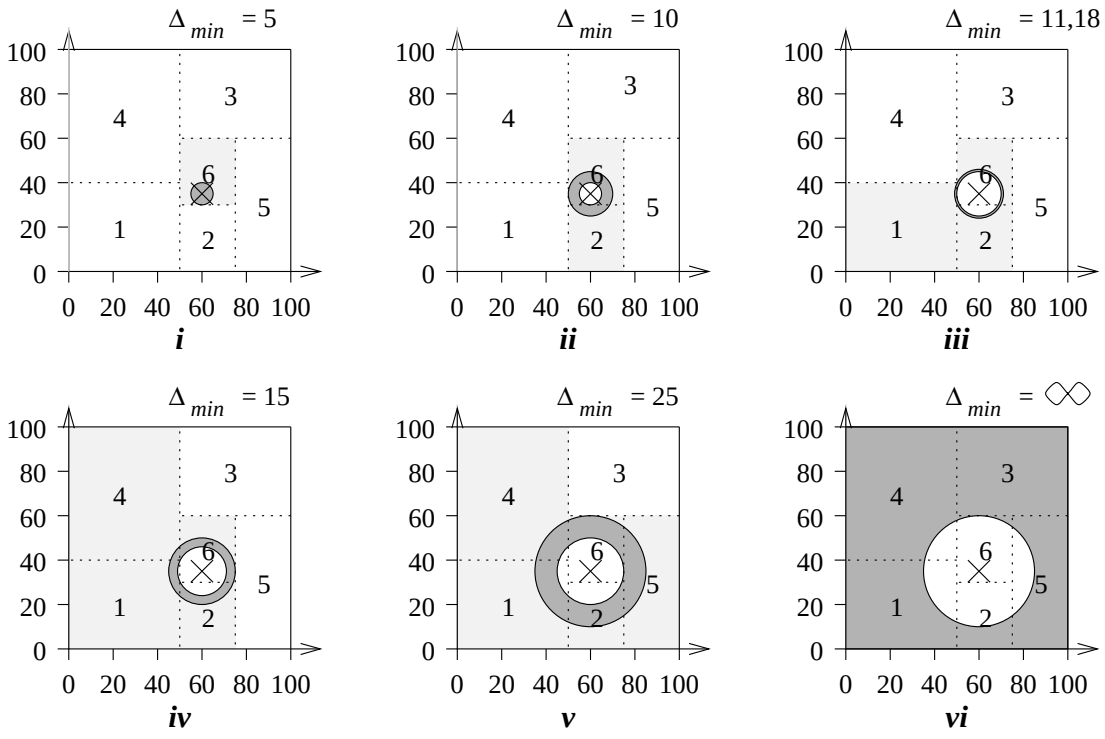
Der Algorithmus für geometrische „Distance-Scan“-Anfragen

Wir benötigen eine Prioritätswarteschlange OPQ, in der die Objekte mit minimalem Abstand zum Anfragepunkt p höchste Priorität haben.

Grober Ablauf des Algorithmus:

1. Suche das Bucket, dessen Bucketregion den Anfragepunkt p enthält.
2. Die in diesem Bucket gespeicherten Objekte werden in OPQ eingefügt.
3. Der minimale Abstand Δ_{min} zwischen dem Punkt p und der nächstgelegenen Kante der Bucketregion wird berechnet.
4. Nun werden solange Objekte aus OPQ entnommen, bis der Abstand des ersten Elementes in OPQ zu p größer ist als Δ_{min}.
5. Nun wird das bisher noch nicht betrachtete Bucket, das am nächsten an p liegt, gesucht und mit Schritt 2. fortgefahren.

Beispiel für den Ablauf des Algorithmus für $p = (60; 35)$:



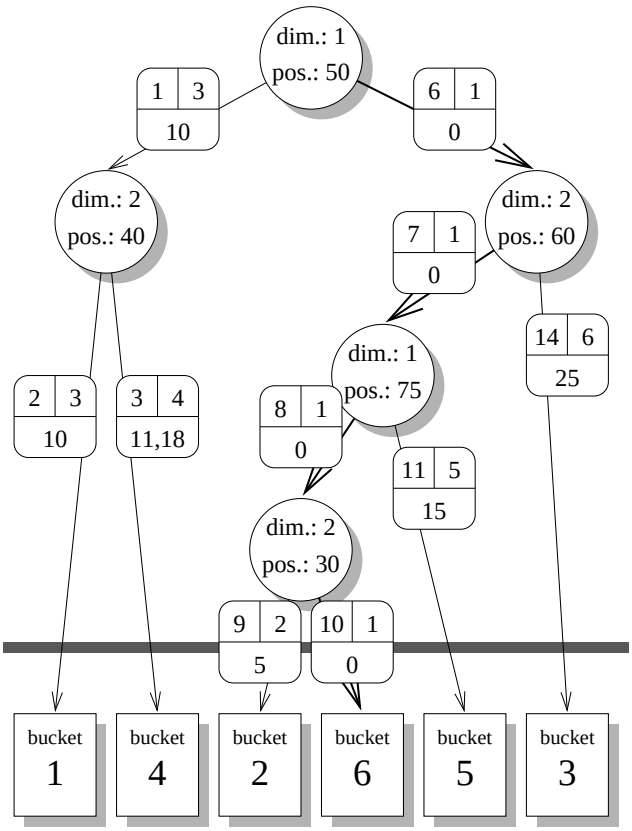
Zusätzlich benötigt man noch eine Hilfsdatenstruktur *NPQ* um die Directoryknoten und Buckets, die später betrachtet werden müssen, zu verwalten.

Zu diesem Zweck wird ein „Handle“ für die Directoryseite oder das Bucket gemeinsam mit der entsprechenden Datenregion in *NPQ* verwaltet.

Die *Datenregion* ist dabei

- für ein Bucket dessen Bucketregion und
- für einen Directoryknoten die Vereinigung der Bucketregionen aller von diesem Knoten aus erreichbaren Buckets.

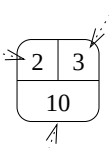
Wie *OPQ*, ist *NPQ* eine Prioritätswarteschlange, in der der Eintrag, dessen Datenregion den geringsten Abstand zu p hat, die höchste Priorität aufweist.



Erklärung:

Nummer des Bearbeitungs-schrittes, in dem das referenzierte Bucket oder der referenzierte Directoryknoten bearbeitet wird

Identifikator

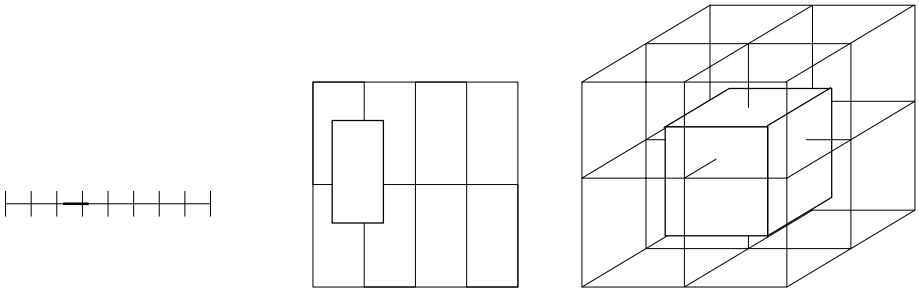


minimaler Abstand zwischen dem Anfragepunkt (60; 35) und der Datenregion des zugehörigen Buckets oder des zugehürigen Directoryknotens (= Priorität)

Was passiert, wenn wir zu höheren Dimensionen übergehen?

Um ein relativ gleich großes Umfeld zu durchsuchen, müssen wir mehr Buckets zugreifen!

Beispiel: Wir wollen ein Umfeld von $\frac{1}{8}$ des Datenraumes durchsuchen:



Es gibt analytische Untersuchungen über die Anzahl der Datensätze, die betrachtet werden müssen, um t Elemente nach dem Abstand sortiert aus OPQ zu entnehmen.

Dabei wird eine Gleichverteilung der Daten und eine Unabhängigkeit der Dimensionen angenommen.

Sei κ die Anzahl der Elemente, die zusätzlich betrachtet (in OPQ eingefügt) werden müssen, dann erhalten wir:

t	$\gamma = 1$	$\gamma = 50$
1	$2{,}2 \leq \kappa$	$3{,}5 \leq \kappa$
2	$3{,}5 \leq \kappa$	$21{,}0 \leq \kappa$
3	$5{,}5 \leq \kappa$	$35{,}1 \leq \kappa$
⋮	⋮	⋮
15	$812{,}1 \leq \kappa$	$3402{,}5 \leq \kappa$
⋮	⋮	⋮
30	$338091{,}2 \leq \kappa$	$1321278{,}4 \leq \kappa$
⋮	⋮	⋮
300	$9,3 \cdot 10^{51} \leq \kappa$	$3,4 \cdot 10^{52} \leq \kappa$

Folge:

für gleichverteilte Punkte ist die Performance höchstens bis zu 10 Dimensionen zufriedenstellend!

$$\Downarrow$$

Darüber müssen ggf. spezielle Charakteristika der Daten (bzw. ihrer Verteilung) ausgenutzt werden!

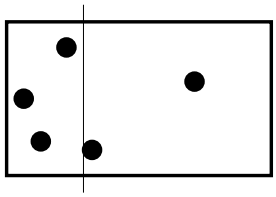
Ansatzpunkte:

- *Abhängigkeiten zwischen Dimensionen*
  $\Rightarrow$ ein Split in einer Dimension impliziert einen Split in anderen Dimensionen
- *Ungleichverteilung*
  Beispiel: im Vektorraummodell $> 90\%$ aller Koordinaten 0
  $\Rightarrow$ Cluster-Splits $\rightarrow$ näheres später

1. kritischer Punkt: die Splitstrategie

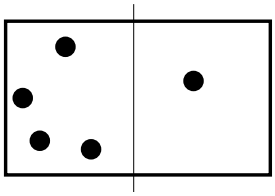
Wie bestimmt man die Splitposition?

1. *datenabhängige Splitstrategien*
   basieren auf den aktuell in dem zu splitenden Bucket gespeicherten Objekten
   Beispiel:
   Man verwendet der Mittelwert über die Objektkoordinaten in der Splitdimension



   Problem: Einfügungen in sortierter Folge
   — das Problem wird aber mit zunehmender Dimensionszahl schwächer

2. *verteilungsabhängige Splitstrategien*
   beruhen auf einer Hypothese über die Verteilung der Objekte
   Beispiel: Gleichverteilung ⇒ splitte in der Mitte



   Problem: es wird eine sinnvolle Verteilungshypothese benötigt
   — dies wird für höhere Dimensionen immer schwieriger

⇓

bei hochdimensionalen Daten sollte man den Mittelwert über die Objektkoordinaten verwenden

Wie soll man die Splitdimension bestimmen?

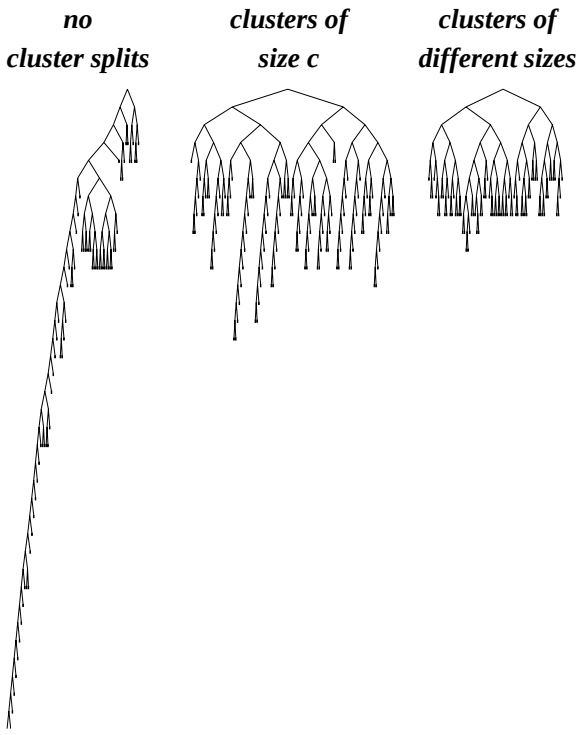
Ziel: vorrangig „selektive“ Dimensionen verwenden

1. *einfache Strategie*
   - Verwende die Dimensionen zyklisch.
   - Verwende eine Dimension dabei aber nur dann, wenn in dem zu splittenden Bucket für die Dimension verschiedene Werte enthalten sind.

2. *Auswahl nach der Varianz*
   - Für sehr hohe Dimensionszahlen und einen Baum mit Höhe < 30 würde die obige Strategie gar nicht alle Dimensionen betrachten.
   - Alternative (z.B. VAMSplit R-tree):
     Nutze die Dimension mit der höchsten Varianz bezogen auf die Werte in dem zu splittenden Bucket.

Im Vektorraummodell:

- Die Objekte können bei einem Split gar nicht 50:50 aufgeteilt werden!
- Interessante Terme treten jeweils nur in weniger als 10 % der Dokumente auf.
- $\Rightarrow$ In jeder Dimension gilt $w_{dk} > 0$ für höchstens 10 % der Punkte.
- $\Rightarrow$ Das Beste, was wir erreichen könnten, wäre eine 90:10 Aufteilung!
- $\Rightarrow$ Der Baum wird recht unbalanciert werden!



linear list containing 80 elements
cut off due to space limitations

Ausweg: Cluster Splits

Betrachte c konsekutive Dimensionen bei einem Split

Speichere ein Objekt im linken Sohn falls gilt

$$\forall k \left(s_{dim} \leq k < \min(s_{dim} + c, t)\right) : w_{dk} = 0$$

und im rechten Sohn falls

$$\exists k \left(s_{dim} \leq k < \min(s_{dim} + c, t)\right) : w_{dk} > 0\,.$$

Für die optimale Clustergröße gilt:

$$\left(1 - P\left(w_{dk} > 0\right)\right)^c = 0.5$$

Folglich:

$$c = \left\lfloor \frac{\ln 0.5}{\ln\left(1 - P\left(w_{dk} > 0\right)\right)} \right\rfloor$$

Die Splitstrategie

Die Splitstrategie betrachtet die Clustergrößen c_i für $i \in \{0, 1, \ldots, 5\}$:

$$c_i = \begin{cases} c & \text{if} \quad i = 0 \\ \left\lceil \frac{c}{3^i} \right\rceil & \text{if} \quad i > 0 \end{cases}$$

Berechne die potentiellen Splitpositionen für Nicht-Clustersplits $s_{pos}^{(k,\diamond)}$ und für Clustersplits $s_{pos}^{(k,c_i)}$:

$$\begin{aligned} s_{pos}^{(k,\diamond)} &= \tfrac{1}{b+1} \cdot \sum_{D \in B} w_{dk} \\ s_{pos}^{(k,c_i)} &= 0 \qquad\qquad \text{for} \quad i \in \{0, 1, \ldots, 5\} \end{aligned}$$

Wir definieren die Homogenität eines Splits nun als die Gleichmäßigkeit der Verteilung der Datensätze.

Dann sollte man unter den geprüften möglichen Kombinationen von Splitdimension, Splitposition und Clustergröße die wählen, für die die Homogenität maximal ist.

Wo greift die Metrik / das Ähnlichkeitsmaß ein?

- Priorität der Objekte in *OPQ*:

  „Abstand“ oder „Ähnlichkeit“ zwischen dem Objekt und dem Anfragepunkt

  – kleiner „Abstand“ = hohe Priorität
  – große „Ähnlichkeit“ = hohe Priorität

  → i. allg. unproblematisch

- Priorität der „Teilbäume“ in *NPQ*:

  minimal möglicher „Abstand“ bzw. maximal mögliche „Ähnlichkeit“ zwischen einem Objekt im Teilbaum und dem Anfragepunkt

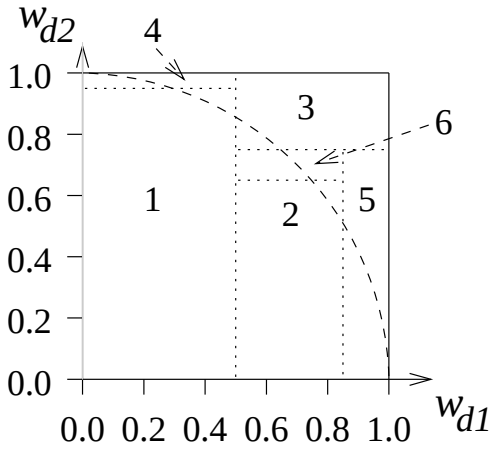
  Beachte: höchste Priorität in *NPQ* dient als Schranke bei der Entnahme aus *OPQ*

  ⇒ Aufgabenstellung: Berechne aus der *Datenregion* des Teilbaumes die Priorität für *NPQ*

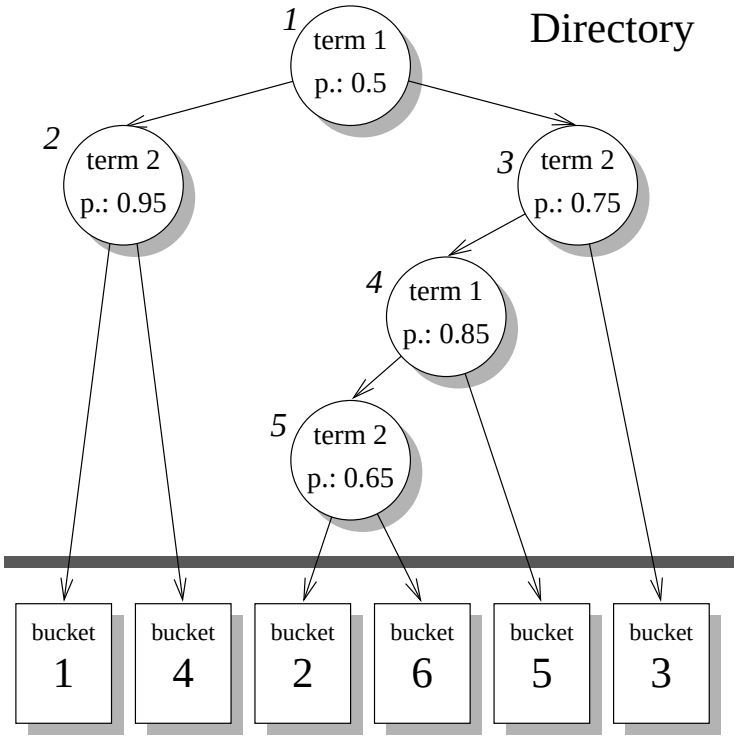
  → kann im konkreten Fall extrem aufwendig sein!

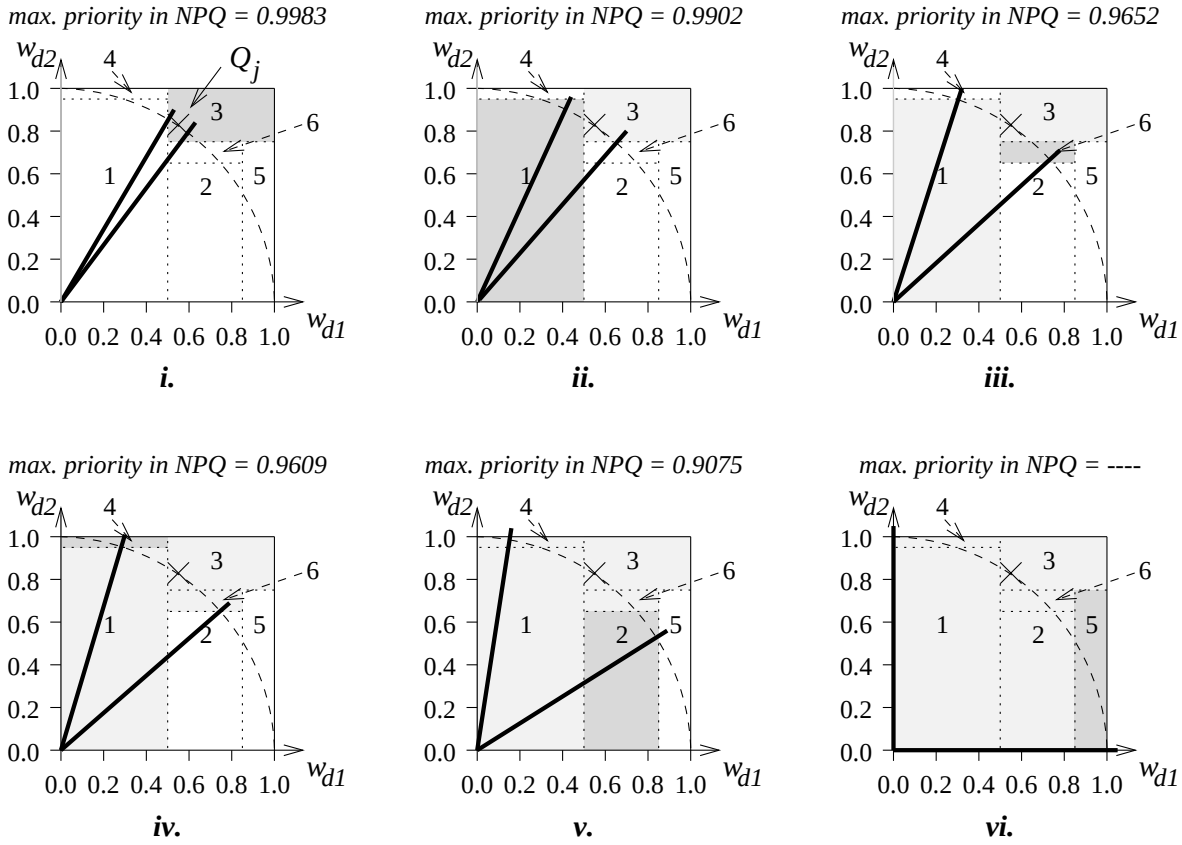
Wie sieht ein LSD-Baum mit Dokumentbeschreibungsvektoren aus?

Aufteilung des Datenraumes:



Directory





Die Prioritäten der Elemente in NPQ

Sei

$$R(\beta) = [\beta_{\lambda_1}, \beta_{\upsilon_1}] \times [\beta_{\lambda_2}, \beta_{\upsilon_2}] \times \ldots \times [\beta_{\lambda_t}, \beta_{\upsilon_t}]$$

die Datenregion eines Buckets oder Directoryknotens β in NPQ:

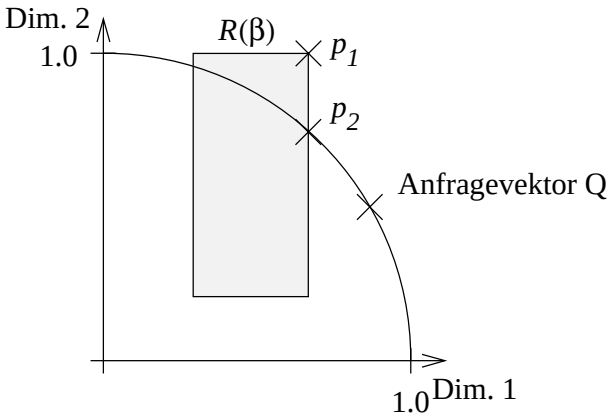
Die Priorität von β in NPQ könnte dann definiert werden durch:

$$SIM\,(Q, R(\beta)) = \max\{sim(Q, D) \mid D \in R(\beta)\}$$

Eine einfache obere Schranke für $SIM\,(Q, R(\beta))$ wäre:

$$SIM'\,(Q, R(\beta)) = \Sigma_{k=1}^{t}\, w_{qk} \cdot \beta_{\upsilon_k}$$

Man könnte aber auch die Normierung der Beschreibungsvektoren ausnutzen, um schärfere Schranken für $SIM\,(Q, R(\beta))$ zu definieren.



Annahme hier (aber nicht notwendig): Anfragevektor ist auch normiert

Skalarprodukt für p_1 deutlich höher als für p_2!

Um eine exakte obere Schranke zu berechnen müßte man ein nichtlineares Optimierungsproblem lösen:

Gesucht wird der Vektor $X = (x_1, x_2, \ldots, x_t)$ mit:

$$\forall k \in \{1, \ldots, t\}: \ \beta_{\lambda_k} \leq x_k \leq \beta_{v_k}$$
$$\Sigma_{k=1}^{t} {x_k}^2 \ = \ 1$$
$$\Sigma_{k=1}^{t} w_{qk} \cdot x_k \ = \ \textbf{Max!}$$

Jedesmal ein solches Optimierungsproblem zu lösen, wenn ein Eintrag in NPQ gemacht wird, wäre viel zu aufwendig!

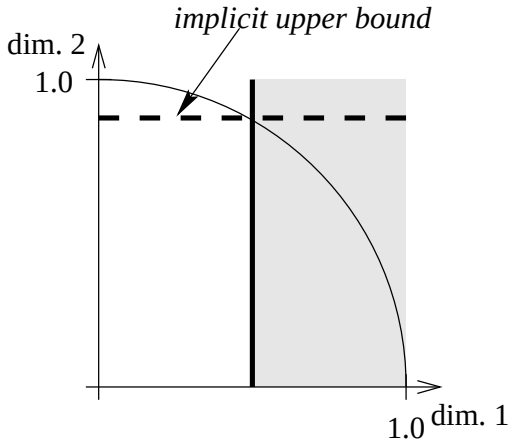
Andererseits bewirken aber zu schwache Schranken für die potentielle Ähnlichkeit

- unnötige Bucketzugriffe
- eine unnötig hohe Anzahl von Elementen in OPQ

Wir verwenden deshalb eine Heuristik, die zwei Aspekte ausnutzt.

1. Aus der Normierung der Beschreibungsvektoren folgt $\sum_{j=1}^{t} w_{dj}^2 = 1$

$\Rightarrow$ eine untere Schranke $\beta_{\lambda_k} > 0$ impliziert eine obere Schranke von $\sqrt{1 - \beta_{\lambda_j}^2}$ in allen anderen Dimensionen ($j \neq k$)



$\beta_{\lambda_1} = 0.5 \Rightarrow \beta'_{v_2} = \sqrt{1 - 0.5^2} \approx 0.87$

Dies können wir nun allgemein für den t-dimensionalen Fall betrachten.

Alle Beschreibungsvektoren liegen auf der Oberfläche der t-dimensionalen Einheitskugel

$\Rightarrow x_k \leq \sqrt{1 - \Sigma_{i \in \{1,\ldots,t\} \wedge i \neq k} \beta_{\lambda_i}{}^2}$

$\Rightarrow$ Wir erhalten eine verschärfte obere Schranke β'_{v_k} anstelle von β_{v_k}:

$$\beta'_{v_k} = \min\left(\beta_{v_k}, \sqrt{1 - \Sigma_{i \in \{1,\ldots,t\} \wedge i \neq k} \beta_{\lambda_i}{}^2}\right)$$

Damit erhalten wir eine verbesserte Version von $SIM'(Q, D(\beta))$:

$$SIM'(Q, D(\beta)) = \Sigma_{k=1}^{t} w_{qk} \cdot \beta'_{v_k}$$

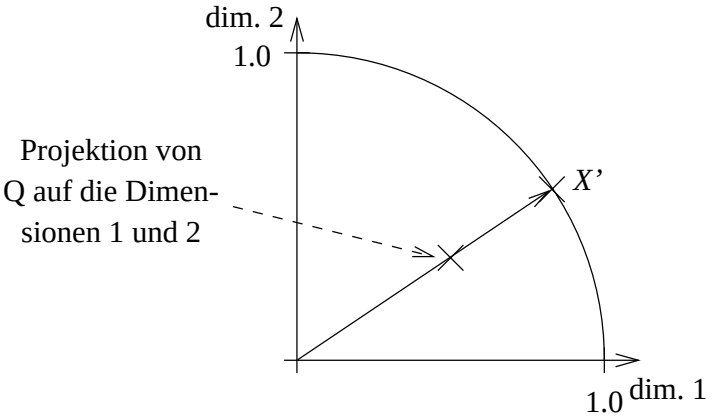
2. $\beta_{\lambda_k} = \beta_{v_k} = 0 \;\Rightarrow\; x_k = 0$

Dies kann wie folgt ausgenutzt werden:

Wir betrachten $Q = (0.5\,;0.331662\,;0.8)$ und nehmen an $\beta_{v_3} = 0$

$\Rightarrow x_3$ nuss Null sein

$\Rightarrow$ Dann können wir ein X', für das $sim(Q, X')$
eine obere Schranke für $SIM\,(Q, R(\beta))$ ist, wie folgt berechnen:



Allgemein ergeben sich die x'_i für die Dimensionen mit $\beta_{v_i} \neq 0$ also durch $\xi \cdot w_{qi}$, wobei ξ der Verlängerungsfaktor ist, mit dem X' auf die Oberfläche der Einheitskugel verlängert wird. ξ bestimmt sich wie folgt:

$$\begin{aligned}
\Sigma_{i\in\{1,\ldots,t\} \text{ mit } \beta_{v_i}\neq 0}\, w_{qi} \cdot x'_i &= 1 \\
\Sigma_{i\in\{1,\ldots,t\} \text{ mit } \beta_{v_i}\neq 0}\, w_{qi} \cdot (\xi \cdot w_{qi}) &= 1 \\
\Sigma_{i\in\{1,\ldots,t\} \text{ mit } \beta_{v_i}\neq 0}\, w_{qi}^2 \cdot \xi &= 1 \\
\xi \cdot \Sigma_{i\in\{1,\ldots,t\} \text{ mit } \beta_{v_i}\neq 0}\, w_{qi}^2 &= 1 \\
\xi &= \frac{1}{\Sigma_{i\in\{1,\ldots,t\} \text{ mit } \beta_{v_i}\neq 0}\, w_{qi}^2}
\end{aligned}$$

Damit erhalten wir:

$$x'_i = \begin{cases} \xi \cdot w_{qi} & \text{falls } \beta_{v_i} \neq 0 \\ 0 & \text{falls } \beta_{v_i} = 0 \end{cases}$$

Als Priorität für die Einträge in NPQ verwenden wir dann

$$SIM'\left(\vec{Q}, D(\beta)\right) = \min\left(\sum_{k=1}^{t} w_{qk} \cdot \beta'_{v_k}\ ,\ sim(\vec{Q}, X')\right)$$

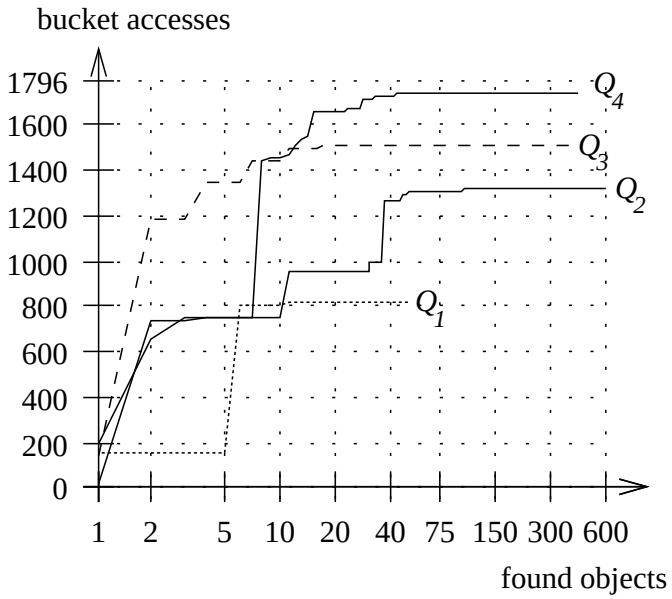
Zusammenfassend nutzen wir

$$\min\left(\sum_{k=1}^{t} w_{qk} \cdot \beta'_{v_k}\ ,\ sim(Q, X')\right)$$

als Priorität für die Einträge in NPQ.

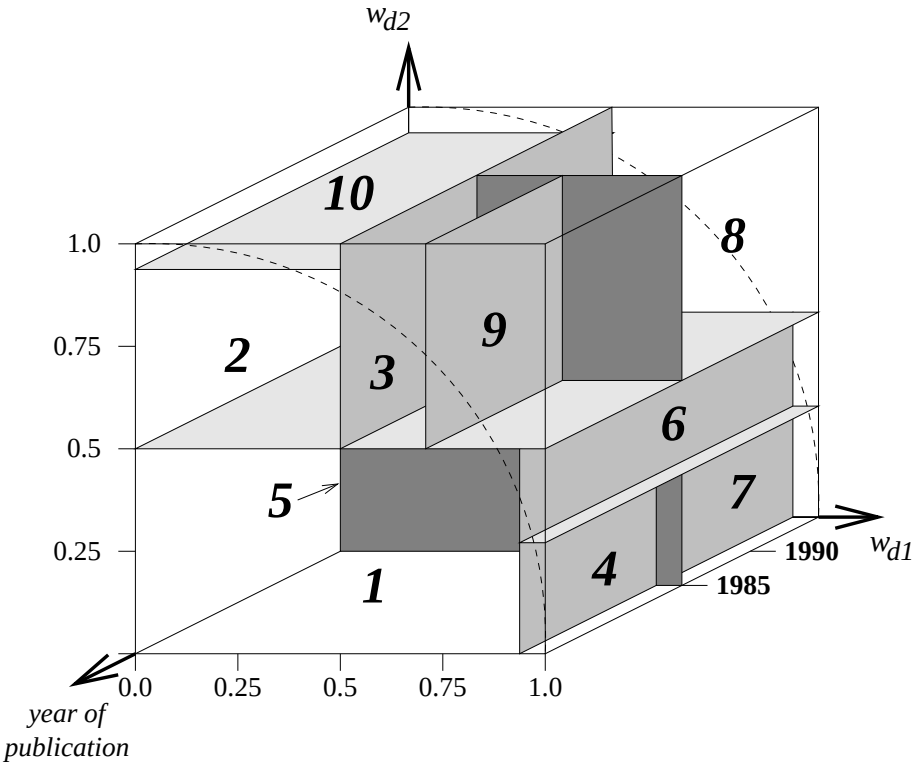
Experimentelle Ergebnisse

- 9135 Verse aus dem Neuen Testament in Deutsch
- Das Vokabular enthielt $t = 1299$ Terme
- Die optimale Clustergröße war $c = 242$
- Bucketauslatung 67,3 %
- 1795 Knoten im Directory
- 1796 Buckets
- Über den ganzen Directorybaum hinweg 112,3 Splits pro Term im Durchschnitt!
- $\Rightarrow$ an jedem der 1795 Splits sind im Durchschnitt $\frac{112{,}3 \times 1299}{1796} = 81,2$ Dimensionen beteiligt.
- 4 Verse wurden als Anfragetexte verwendet



Letztlich ergibt sich somit ein ähnliches Bild, wie bei den invertierten Listen.

Idee der Kombination mit zusätzlichen Bedingungen



7.3 Relevance-Feedback

- Die Anfrage ist nur eine unvollkommene Beschreibung des Informationswunsches!
- $\Rightarrow$ weitere Quellen zur Verbesserung des Ergebnisses nutzen!
- Eine möglische Quelle:
  Die Beurteilung eines ersten Ergebnisses durch den Fragesteller.

Der Anfragevektor sollte dann

- von den Beschreibungsvektoren der gefundenen *nicht* relevanten Dokumente weg
- in Richtung der gefundenen relevanten Dokumente verschoben werden.

In der Literatur wurden zahlreiche Formeln vorgestellt, von denen wir nur die in [SB90] verglichenen hier exemplarisch darstellen wollen.

Wir gehen dabei davon aus, dass die bisher vom Anfragesteller betrachteten Dokumente (Feedbackmenge) in zwei Mengen F^+ und F^- zerfallen.

F^+ enthält die als relevant eingestuften Dokumente und F^- die als nicht relevant eingestuften (ggf. können auch Dokumente in eine dritte Menge $F^?$ eingeordnet werden).

- **Ide (dec hi)**
  Die Beschreibungsvektoren der Dokumente aus F^+ werden hier zum Anfragevektor hinzuaddiert.
  Ferner wird der Beschreibungsvektor $\mathcal{D}_1^{\overline{rel}}$ des von der bisherigen Anfrage am höchsten eingeschätzten Dokumentes aus F^- von dem entstehenden Anfragevektor abgezogen.
  Damit wird der Anfragevektor von diesem nicht relevanten Dokument weg zu den relevanten Dokumenten hin verschoben.

$$\mathcal{Q}_{neu} = \mathcal{Q}_{alt} + \left(\sum_{D \in F^+} \mathcal{D} \right) - \mathcal{D}_1^{\overline{rel}}$$

- **Ide (regular)**

  Hier werden im Gegensatz zu Ide (dec hi) alle Dokumente aus F^- berücksichtigt.

$$\mathcal{Q}_{neu} = \mathcal{Q}_{alt} + \left(\sum_{D \in F^+} \mathcal{D}\right) - \left(\sum_{D \in F^-} \mathcal{D}\right)$$

- **Rocchio**

  Man gewichtet den Einfluss der relevant erachteten Dokumente mit β und den der nicht relevant erachteten Dokumente mit α ($\beta + \alpha = 1$):

  Es wird letztlich der gewichtete Zentroid der relevanten Dokumente addiert und der gewichtete Zentroid der nicht relevanten Dokumente subtrahiert.

$$\mathcal{Q}_{neu} = \mathcal{Q}_{alt} + \beta \cdot \left(\sum_{D \in F^+} \frac{\mathcal{D}}{|F^+|}\right) - \alpha \cdot \left(\sum_{D \in F^-} \frac{\mathcal{D}}{|F^-|}\right)$$

- **Probabilistic conventional**

  Die folgenden Formeln werden später beim Probabilistischen IR noch genauer betrachtet.

  r_k = Anzahl der relevanten Dokumente, die Term k enthalten

  n_k = Anzahl der Dokumente, die Term k enthalten ohne Rücksicht auf die Relevanz

  N = Anzahl der Dokumente in der Feedbackmenge

  (r_k, n_k und N verstehen sich hier relativ zur Feedbackmenge $F = F^+ \cup F^- \cup F^?$)

  Die Dokumentbeschreibungsvektoren werden hier als binär unterstellt:
  $X = \{x_1, x_2, \ldots, x_t\}$ mit $x_i \in \{0; 1\}$

$$w_{q_{new}k} = \log \left[\frac{p_k(1-u_k)}{u_k(1-p_k)}\right]$$

$$p_k = P(x_k = 1|\text{rel}) = \frac{r_k+0.5}{|F^+|+1.0}$$

$$u_k = P(x_k = 1|\text{nonrel}) = \frac{n_k-r_k+0.5}{N-|F^+|+1.0}$$

- **Probabilistic adjusted derivation**

$$w_{q_{new}k} = \log\left[\frac{p'_k(1-u'_k)}{u'_k(1-p'_k)}\right]$$

$$p'_k = P(x_k = 1|\text{rel}) = \frac{r_k + \frac{n_k}{N}}{|F^+|+1.0}$$

$$u'_k = P(x_k = 1|\text{nonrel}) = \frac{n_k - r_k + \frac{n_k}{N}}{N-|F^+|+1.0}$$

- **Probabilistic adjusted derivation revised**

$$w_{q_{new}k} = \log\left[\frac{p'_k(1-u'_k)}{u'_k(1-p'_k)}\right]$$

$$p'_k = P(x_k = 1|\text{rel}) = \frac{(r_k+3) + \frac{n_k}{N}}{(|F^+|+3)+1.0}$$

$$u'_k = P(x_k = 1|\text{nonrel}) = \frac{n_k - (r_k+3) + \frac{n_k}{N}}{N-(|F^+|+3)+1.0}$$

Experimentelle Ergebnisse

- Es wurden insgesamt 72 Verfahren verglichen. Die Ränge der Verfahren beziehen sich auf den Rang unter diesen 72.
- Die Precision-Werte sind Mittelwerte über die Recall-Punkte 0.75, 0.5 und 0.25.
- Zum Relevanz-Feedback wurden die 15 Dokumente verwendet, die von der initialen Suche (nach klassischem Vektorraummodell) als „relevanteste“ ermittelt wurden.

  Diese 15 Dokumente wurden weder bei der initialen noch bei den nachfolgenden Suchen bei der Berechnung von Recall und Precision berücksichtigt.
- *expand by most common terms* berücksichtigt jeweils nur die Terme bei der Reformulierung, die in vielen relevanten Dokumenten vorkommen.

  (Ein pragmatischer Vorteil dieses Vorgehens:

  im Anfragevektor sind weniger Komponenten $\neq 0$, was z.B. bei der Anwendung des Buckley&Lewit-Algorithmus vorteilhaft ist.)

Relevance Feedback Method		CACM 3204 docs 64 queries	CISI 1460 docs 112 queries	CRAN 1397 docs 225 queries	INSPEC 12684 docs 84 queries	MED 1033 docs 30 queries	Average
Initial Run							
		.1459	.1184	.1156	.1368	.3346	
Ide (dec hi)							
expand by	Rank	1	2	6	1	1	2.2
all terms	Precision	.2704	.1742	.3011	.2140	.6305	
	Improvement	+86%	+47%	160%	+56%	+88%	+87%
expand by most	Rank	4	1	13	2	3	4.6
common terms	Precision	.2479	.1924	.2498	.1976	.6218	
	Improvement	+70%	+63%	+116%	+44%	+86%	+76%
Ide (regular)							
expand by	Rank	7	18	15	4	2	9.2
all terms	Precision	.2241	.1550	.2508	.1936	.6228	
	Improvement	+66%	+31%	+117%	+42%	+86%	+68%
expand by most	Rank	17	5	17	17	4	12
common terms	Precision	.2179	.1704	.2217	.1808	.5980	
	Improvement	+49%	+44%	+92%	+32%	+79%	+59%
Rocchio (standard $\beta = .75$, $\alpha = .25$)							
expand by	Rank	2	39	8	14	17	16
all terms	Precision	.2552	.1404	.2955	.1821	.5630	
	Improvement	+75%	+19%	+156%	+33%	+68%	+70%
expand by most	Rank	3	12	12	10	24	12.2
common terms	Precision	.2491	.1623	.2534	.1861	.5297	
	Improvement	+71%	+37%	+119%	+36%	+55%	+64%

Relevance Feedback Method		CACM 3204 docs 64 queries	CISI 1460 docs 112 queries	CRAN 1397 docs 225 queries	INSPEC 12684 docs 84 queries	MED 1033 docs 30 queries	Average
Initial Run							
		.1459	.1184	.1156	.1368	.3346	
Probabilistic adjusted revised derivation							
expand by	Rank	11	36	3	32	5	17.4
all terms	Precision	.2289	.1436	.3108	.1621	.5972	
	Improvement	+57%	+21%	+169%	+19%	+78%	+69%
expand by most	Rank	14	10	18	9	14	13
common terms	Precision	.2224	.1634	.2120	.1876	.5643	
	Improvement	+52%	+38%	+83%	+37%	+69%	+56%
Conventional Probabilistic							
expand by	Rank	18	56	1	55	13	28.6
all terms	Precision	.2165	.1272	.3117	.1343	.5681	
	Improvement	+48%	+7%	+170%	-2%	+70%	+59%
expand by most	Rank	12	4	11	19	8	10.8
common terms	Precision	.2232	.1715	.2538	.1782	.5863	
	Improvement	+53%	+45%	+120%	+30%	+75%	+65%

Zusammenfassung Relevanz-Feedback

- (Fast) unabhängig vom Verfahren liefert Relevanz-Feedback eine deutliche Verbesserung der Retrievalqualität.
- Es erscheint dabei aus algorithmischen Gründen sinnvoll, nur die Terme bei der Anfrageverbesserung zu betrachten, die in mehreren relevanten Dokumenten auftreten.
- Relevanz-Feedback bringt eine wesentlich größere Steigerung der Retrievalqualität als ein Feintuning an den ursprünglichen Formeln.

  Was wenig verwunderlich ist, weil ja zusätzliche Informationen über den Informationswunsch in die Anfragebearbeitung mit einbezogen werden.

7.4 Varianten

Es gibt in der Literatur (insbesondere in den Konferenzbänden der SIGIR [Hea99, Cro98]) eine Vielzahl von Detailvorschlägen zur Verbesserung der $tf \cdot idf$-Formel.

Wir wollen hier exemplarisch nur einen herausgreifen, der sich mit der Normierung der Beschreibungsvektoren zu den Dokumenten beschäftigt [SBM96].

Das Problem:

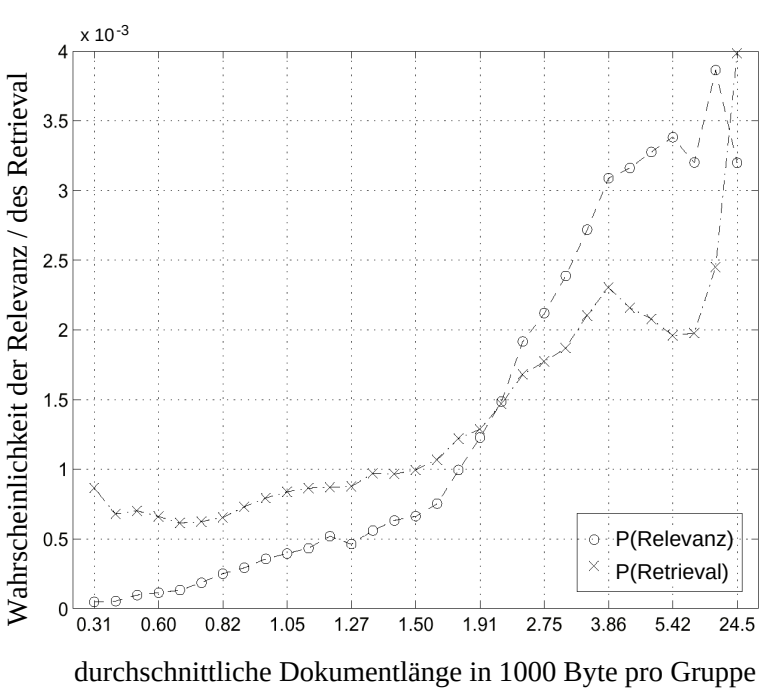
- Kurze Dokumente enthalten recht wenige Terme.

  Jeder einzelne Term, der in einem kurzen Dokument vorkommt hat daher einen relativ hohen w_{dk} Wert.
- Lange Dokumente enthalten typischerweise viele verschiedene Terme.

  Durch die Normierung hat dann jeder dieser Terme einen relativ kleinen Wert w_{dk}.
- Werden in einer Anfrage nur wenige Terme vorgegeben, werden die Dokumente zuerst ermittelt, die für diese Terme hohe w_{dk} Werte haben.

  Das werden eher kurze als lange Dokumente sein!

Beipiel: TREC CD's 1 und 2 (741856 Dokumente) mit Topics 151 bis 200:



Effekt:

- Für kurze Dokumente ist die Wahrscheinlichkeit, dass sie im Ergebnis einer Anfrage sind höher als die Wahrscheinlichkeit, dass sie für eine Anfrage relevant sind!
- Für lange Dokumente ist es genau umgekehrt!

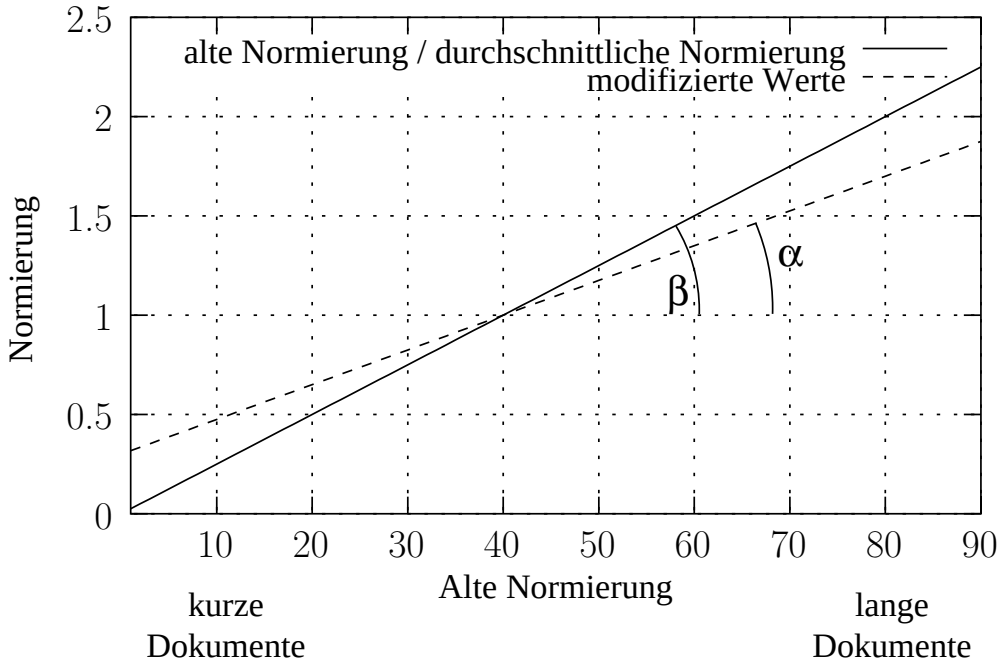
Man sollte also

- die Normierung $\sqrt{\sum_{i=1}^{t} \left(tf_{di} \cdot \log \frac{N}{n_i}\right)^2}$ für lange Dokumente etwas abschwächen und damit die w_{dk} tendenziell etwas anheben und
- die Normierung für kurze Dokumente etwas verschärfen und damit die w_{dk} tendenziell etwas senken.

Dazu wird nun folgende Formel verwendet:

$$(1.0 - slope) + slope \times \frac{old\ normalization}{average\ old\ normalization}$$

Folgende Abbildung verdeutlicht den Effekt für $slope = 0.7$ und $average\ old\ normalization = 40$:



Das Verhältnis zwischen α und β gibt dabei den Wert von *slope* wieder: $slope = \frac{\alpha}{\beta}$

Der beste Wert für *slope* muss nun empirisch bestimmt werden.

Für die CD's 1 und 2 von TREC mit 741856 Dokumenten ergab sich eine *average old normalization* von 13.36.

Mit den Topics 151 bis 200 ergaben sich folgende Werte, wobei

- die erste Zahl für die Anzahl der über alle Topics hinweg gefundenen relevanten Dokumente steht (von 9805) und
- die zweite Zahl für die durchschnittliche Precision.

	Pivoted Cosine Normalization Slope				
Cosine	0.60	0.65	0.70	**0.75**	0.80
6526	6342	6458	6574	**6629**	6671
0.2840	0.3024	0.3097	0.3144	**0.3171**	0.3162
Improvement	+6.5%	+9.0%	+10.7%	**+11.7%**	+11.3%

Zusammenfassung

Das Verfahren versucht durch eine modifizierte Normierung die Wahrscheinlichkeit der Relevanz und die Wahrscheinlichkeit des Retrieval anzugleichen.

Dabei sind folgende Aspekte zu beachten:

- Durch die modifizierte Normierung gilt nicht mehr $w_{dk} \leq 1$.
  Alle Zugriffstrukturen und -algorithmen, die dies ausnutzen, müssen entsprechend angepaßt werden.
- Der Wert für *slope* muss empirisch ermittelt werden.
  Ob es eine allgemein sinnvolle Belegung dafür gibt, ist offen.

Kapitel 8

Probabilistisches Information Retrieval

- Das BIR-Modell (binary independence retrieval)
- Dokumenten-Indexierung durch probabilistisches Lernen

8.1 Das BIR-Modell („binary independence retrieval“)

Ziel:

- Berechnung der Wahrscheinlichkeit, dass ein Dokument D für eine Anfrage Q relevant ist: $P(D \in R^+(Q))$
- Da wir beim IR immer auf Darstellungen arbeiten müssen, erhalten wir: $P(D' \in R^+(Q) \mid \beta(D') = \beta(D))$

Das klassische Modell des Probabilistischen IR wurde 1976 von Roberston und Sparck Jones [RJ76] vorgestellt.

Dieses Modell wird auch als „binary independence retrieval“-Modell (BIR) bezeichnet.

Basisidee des BIR-Modells (in Anlehnung an [BYRN99], S. 30ff.):

- für eine Anfrage eines Benutzers gibt es eine Menge von Dokumenten, die genau die relevanten Dokumente beinhaltet und damit die ideale Antwort darstellt.
- Wenn wir eine genaue Beschreibung dieser idealen Antwort hätten, wäre es kein Problem ihre Elemente zu ermitteln.

⇒ Anfrageprozess ≈ Prozess der Bestimmung der Eigenschaften der idealen Anwort

- Problem: wir kennen diese Eigenschaften nicht
- Wir wissen nur, dass es Indexterme gibt, deren Semantik wir verwenden sollten/könnten, um diese Eigenschaften zu charakterisieren.
- Eigenschaften a priori nicht bekannt ⇒ es muss ein initialer Schritt unternommen werden, der mit einer groben Näherung der Eigenschaften ein erstes Ergebnis liefert.
- Dieses Ergebnis kann dann verwendet werden, um eine vorläufige probabilistische Beschreibung der Eigenschaften der idealen Antwort zu erhalten.
- Im Zusammenspiel mit dem Anfragenden kann dann ein iterativer Prozess stattfinden, in dem die probabilistische Beschreibung verfeinert wird.

Ablauf des iterativen / interaktiven Prozesses:

- Der Anfragesteller betrachtet die ermittelten Dokumente und entscheidet (zumindest für die ersten von ihnen) welche tatsächlich relevant sind, und welche nicht.
- Das System nutzt dann diese Information um die Eigenschaften der idealen Antwort genauer zu beschreiben.
- Durch Wiederholung dieses Vorgangs wird erwartet, dass sich die Beschreibung der Eigenschaften an die Eigenschaften der tatsächlichen idealen Antwort annähert.

Dabei muss man aber immer im Auge behalten, dass zunächst eine initiale Beschreibung der idealen Antwort benötigt wird.

Dem probabilistischen Modell liegen dabei die folgenden Annahmen zugrunde:

- Für eine Anfrage Q und ein Dokument D aus der Dokumentenkollektion versucht das probabilistische Modell die Wahrscheinlichkeit abzuschätzen, dass D im Hinblick auf Q für relevant erachtet wird.
- Das Modell unterstellt, dass diese Wahrscheinlichkeit nur von der Anfrage- und der Dokumentenrepräsentation abhängig ist.
- Ferner nimmt das Modell an, dass es eine ideale Antwort auf die Anfrage Q gibt, die alle aus der Sicht des Anfragenden für Q relevanten Dokumente enthält.
- Diese ideale Antwort bezeichnen wir als R_Q^+.

Für eine Anfrage Q ordnet das probabilistische Modell jedem Dokument D als Ähnlichkeitsmaß $\frac{P(D \text{ ist relevant im Hinblick auf } Q)}{P(D \text{ ist nicht relevant im Hinblick auf } Q)}$ zu.

Dieses Maß entspricht der Chance des Dokuments D relevant für Q zu sein.

Einige Definitionen:

- Die Gewichte in den Beschreibungsvektoren seien alle binär, d.h.: $\forall_{i,D} : w_{di} \in \{0;1\}$ und $\forall_{i,Q} : w_{qi} \in \{0;1\}$
- R_Q^- sei das Komplement zu R_Q^+.
- $P(D' \in R_Q^+ | \beta(D') = \beta(D))$ sei die Wahrscheinlichkeit, das ein Dokument D', dessen Repräsentation $\beta(D')$ der Repräsentation $\beta(D)$ von Dokument D entspricht, relevant ist.

Dann können wir die Ähnlichkeit eines Dokumentes D mit einer Anfrage Q formulieren als:

$$sim(D,Q) = \frac{P(D' \in R_Q^+ | \beta(D') = \beta(D))}{P(D' \in R_Q^- | \beta(D') = \beta(D))}$$

Dies können wir mit Hilfe des Theorems von Bayes vereinfachen.

Das Theorem von Bayes:

$$\begin{aligned}
& P(a|b) = \frac{P(a \wedge b)}{P(b)} && (1)\\
\Leftrightarrow \quad & P(b|a) = \frac{P(a \wedge b)}{P(a)} &&\\
\Rightarrow \quad & P(a \wedge b) = P(a) \cdot P(b|a) && (2)\\
\overset{(2)in(1)}{\Rightarrow} \quad & P(a|b) = \frac{P(a) \cdot P(b|a)}{P(b)} &&
\end{aligned}$$

Wir erhalten:

$$\begin{aligned}
sim(D,Q) &= \frac{P(D' \in R_Q^+) \cdot P(\beta(D') = \beta(D) | D' \in R_Q^+)}{P(\beta(D') = \beta(D))} \cdot \frac{P(\beta(D') = \beta(D))}{P(D' \in R_Q^-) \cdot P(\beta(D') = \beta(D) | D' \in R_Q^-)} \\
&= \frac{P(D' \in R_Q^+) \cdot P(\beta(D') = \beta(D) | D' \in R_Q^+) \cdot P(\beta(D') = \beta(D))}{P(\beta(D') = \beta(D)) \cdot P(D' \in R_Q^-) \cdot P(\beta(D') = \beta(D) | D' \in R_Q^-)} \\
&= \frac{P(D' \in R_Q^+) \cdot P(\beta(D') = \beta(D) | D' \in R_Q^+)}{P(D' \in R_Q^-) \cdot P(\beta(D') = \beta(D) | D' \in R_Q^-)}
\end{aligned}$$

Weil $\frac{P(D' \in R_Q^+)}{P(D' \in R_Q^-)}$ für alle Dokumente gleich ist, können wir dies vereinfachen zu:

$$sim(D,Q) \sim \frac{P(\beta(D') = \beta(D) | D' \in R_Q^+)}{P(\beta(D') = \beta(D) | D' \in R_Q^-)}$$

Bezieht man sich auf die einzelnen Terme (= Dimensionen), so kann dies auch geschrieben werden als:

$$sim(D,Q) \sim \frac{\left(\forall_{\{i|w_{di}=1\}} : P(w_{d'i} = 1 | D' \in R_Q^+)\right) \wedge \left(\forall_{\{i|w_{di}=0\}} : P(w_{d'i} = 0 | D' \in R_Q^+)\right)}{\left(\forall_{\{i|w_{di}=1\}} : P(w_{d'i} = 1 | D' \in R_Q^-)\right) \wedge \left(\forall_{\{i|w_{di}=0\}} : P(w_{d'i} = 0 | D' \in R_Q^-)\right)}$$

Wenn wir nun die Unabhängigkeit der Indexterme untereinander unterstellen, kann dies geschrieben werden als:

$$sim(D,Q) \sim \frac{\left(\Pi_{\{i|w_{di}=1\}} P(w_{d'i} = 1 | D' \in R_Q^+)\right) \cdot \left(\Pi_{\{i|w_{di}=0\}} P(w_{d'i} = 0 | D' \in R_Q^+)\right)}{\left(\Pi_{\{i|w_{di}=1\}} P(w_{d'i} = 1 | D' \in R_Q^-)\right) \cdot \left(\Pi_{\{i|w_{di}=0\}} P(w_{d'i} = 0 | D' \in R_Q^-)\right)}$$

Wenn wir

- von diesem Ausdruck den Logarithmus nehmen,
- beachten, dass $P(w_{d'i} = 1 | D' \in R_Q^+) + P(w_{d'i} = 0 | D' \in R_Q^+) = 1$ gilt und
- Faktoren weglassen, die für alle Dokumente im Hinblick auf die gleiche Anfrage gleich sind,

dann erhalten wir hieraus:

$$sim(D,Q) \sim \sum_{\{i|w_{qi}=w_{di}=1\}} \left(\log \frac{P(w_{d'i} = 1 | D' \in R_Q^+)}{1 - P(w_{d'i} = 1 | D' \in R_Q^+)} + \log \frac{1 - P(w_{d'i} = 1 | D' \in R_Q^-)}{P(w_{d'i} = 1 | D' \in R_Q^-)} \right)$$

Initial müssen wir nun ohne jegliche Vorabinformation grobe Schätzungen für $P(w_{d'i} = 1|D' \in R_Q^+)$ und $P(w_{d'i} = 1|D' \in R_Q^-)$ machen. Vorgeschlagen wurde hierzu:

- $P(w_{d'i} = 1|D' \in R_Q^+)$ sei für alle Terme i gleich und zwar z.B. 0.5

$$\Rightarrow P(w_{d'i} = 1|D' \in R_Q^+) = 0.5$$

- die Verteilung der Indexterme über den nicht relevanten Dokumenten entspreche der Verteilung über allen Dokumenten.

$$\Rightarrow P(w_{d'i} = 1|D' \in R_Q^-) = \frac{n_i}{N}$$

Wir erhalten somit (unter der Voraussetzung dass $N \gg n_i$ gilt):

$$sim(D,Q) \sim \sum_{\{i|w_{qi}=w_{di}=1\}} \left(\log \frac{0.5}{0.5} + \log \frac{1 - \frac{n_i}{N}}{\frac{n_i}{N}} \right) = \sum_{\{i|w_{qi}=w_{di}=1\}} \log \frac{N - n_i}{n_i}$$

Haben wir mit dieser Funktion ein erstes Ergebnis ermittelt, so können wir mit oder ohne Unterstützung durch den Anfragenden unser Ergebnis fortentwickeln.

Zunächst das Szenario ohne Unterstützung durch den Anfragenden:

- Die initiale Formel habe eine Ergebnismenge ergeben, deren r am höchsten gerankte Elemente wir als Menge V bezeichnen wollen.
- Wenn wir diese Menge als repräsentative Stichprobe von R_Q^+ auffassen, dann erhalten wir (man beachte dass per Definition $|V| = r$ gilt):

$$P(w_{d'i} = 1|D' \in R_Q^+) = \frac{|\{D' \in V|w_{d'i} = 1\}|}{|V|}$$

$$P(w_{d'i} = 1|D' \in R_Q^-) = \frac{n_i - |\{D' \in V|w_{d'i} = 1\}|}{N - |V|}$$

- Mit diesen Formeln können wir das Ergebnis jetzt über mehrere Iterationen hinweg verbessern.

Typischerweise verwendet man dabei aber nicht V, sondern eine von dem Anfragenden ausgewählte Menge von relevanten Dokumenten.

Dazu sucht der Anfragende aus der initialen Ergebnismenge die r am höchsten gerankten Dokumente, die er für relevant hält aus.

Wir können die Menge dieser Dokumente z.B. als V^* bezeichnen.

In den obigen Formeln verwenden wir dann V^* statt V.

Da die obigen Formeln bei kleinen Mengen V^*, bzw. V, Probleme bereiten, verwendet man häufig auch

$$P(w_{d'i} = 1 | D' \in R_Q^+) = \frac{|\{D' \in V | w_{d'i} = 1\}| + 0.5}{|V| + 1}$$

$$P(w_{d'i} = 1 | D' \in R_Q^-) = \frac{n_i - |\{D' \in V | w_{d'i} = 1\}| + 0.5}{N - |V| + 1}$$

oder

$$P(w_{d'i} = 1 | D' \in R_Q^+) = \frac{|\{D' \in V | w_{d'i} = 1\}| + \frac{n_i}{N}}{|V| + 1}$$

$$P(w_{d'i} = 1 | D' \in R_Q^-) = \frac{n_i - |\{D' \in V | w_{d'i} = 1\}| + \frac{n_i}{N}}{N - |V| + 1}$$

Der Vorteil des probablilistischen Modells besteht in der Theorie darin, dass die Dokumente absteigend nach ihrer Relevanzwahrscheinlichkeit sortiert werden.

Eine Implementierung ist ohne größere Probleme mit invertierten Listen möglich.

Nachteile:

- Die Notwendigkeit ein initiales Ergebnis aufgrund sehr grober Annahmen zu berechnen.
- Die Verwendung binärer Werte in den Beschreibungsvektoren.
- Die Unabhängigkeitsannahme bezüglich der Indexterme.

Ob das probabilistische Modell bessere Ergebnisse liefert als das Vektorraummodell ist nicht ganz klar.

Allerdings legen sehr ausführliche Tests von Salton und Buckley nahe, dass das Vektorraummodell für reale Kollektionen bessere Ergebnisse liefert.

[Man kann allerdings auch das probabilistische Modell zum Relevanz-Feedback im Rahmen des Vektorraummodells einsetzen (siehe Abschnitt 7.3).]

8.2 Dokumenten-Retrieval als probabilistisches Lernen [FB91]

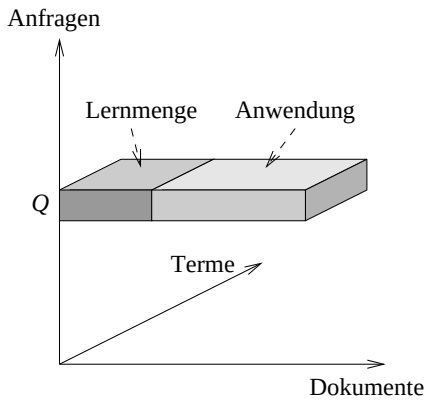
Basiert auf drei Konzepten:

- Abstraktion von spezifischen Termen und Dokumenten
  Beim BIR-Modell wurde im Bezug auf eine konkrete Anfrage „gelernt“.
- Flexibilität der Repräsentation
  Beim BIR-Modell war die Repräsentation festgelegt.
- Probabilistisches Lernen zur Gewinnung der Indexierungsgewichte

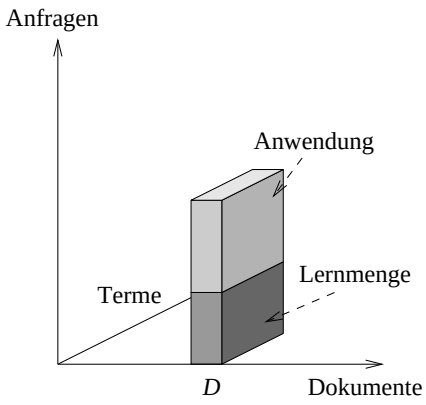
Man unterscheidet nun bei der Indexierung die beiden Schritte:

- Beschreibung (description step) und
- Entscheidung (decision step).

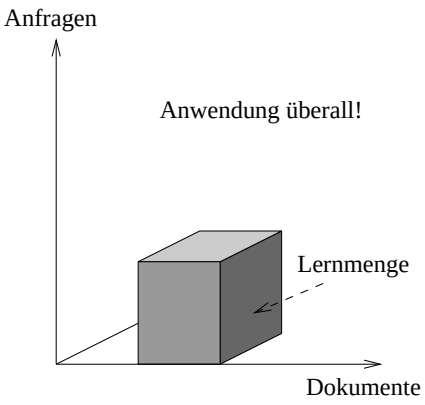
Die Situation kann wie folgt beschrieben werden:



anfragebezogenes Lernen (z.B. BIR-Modell)



dokumentenbezogenes Lernen (z.B. BII-Modell)



Lernen auf der Basis von allgemeinen Eigenschaften (Darmstädter Indexierungsansatz)

Der Ablauf kann dann grob wie folgt beschrieben werden:

- Man wähle einige quantifizierbare, die Relevanz eines Dokumentes D in Relation zu einer Anfrage Q bestimmende Eigenschaften aus.

  Beispiele:

  – die Anzahl der übereinstimmenden Terme
  – die Anzahl der Terme aus der Anfrage, die im Titel des Dokumentes auftreten
  – die Anzahl der Terme im Dokument
  – die Vorkommenshäufigkeit der 3 „wichtigsten“ gemeinsamen Terme im Dokument
  – ...

- Man fasse die Werte für Dokument D und Anfrage Q zu einem Vektor $\vec{V}_{DQ}$ zusammen (dieser Vektor wird auch als Relevanzbeschreibung bezeichnet).
- Man definiere eine Skala für die Relevanz eines Dokumentes (z.B. $0 \ldots 10$).
- Man bestimme für eine Menge von Dokumenten und Anfragen manuell die Relevanzwerte $\mathcal{R}$.

- Man bestimme eine Funktion $\mathcal{F}$, die
  - aus der Menge der Beschreibungsvektoren in das Intervall $[0; 10]$ abbildet und
  - für die gegebene Stichprobe die Abweichungsquadrate minimiert.
- Nun kann man $\mathcal{F}$ auf beliebige Kombinationen aus Anfrage und Dokument in der Kollektion anwenden.
- Man erhält so für eine beliebige Anfrage ein Ranking der Dokumente.
- Das Ranking kann durch Weiterentwicklung von $\mathcal{F}$ weiter verbessert werden.

Man beachte, dass die Relevanzbeschreibung folgende Arten von Eigenschaften enthalten kann:

- Eigenschaften, die sich auf die Beziehung von D und Q beziehen,
- Eigenschaften, die sich nur auf D beziehen, und
- Eigenschaften, die sich nur auf Q beziehen.

Beispiel für die Komponenten einer 12-elementigen Relevanz-Beschreibung $\vec{V}_{DQ}$:

element	description
v_1	# descriptors common to query and document
v_2	log(# descriptors common to query and document)
v_3	highest indexing weight of a common descriptor
v_4	lowest indexing weight of a common descriptor
v_5	# common descriptors with weight ≥ 0.15
v_6	# non-common descriptors with weight ≥ 0.15
v_7	# descriptors in the document with weight ≥ 0.15
v_8	log Σ (indexing weights of common descriptors)
v_9	log(# descriptors in the query)
v_{10}	log(min(size of output set, 100))
v_{11}	= 1, if size of output set > 100
v_{12}	= 1, if request about nuclear physics

Kapitel 9

Weitere IR-Modelle

- Das verallgemeinerte Vektorraummodell
- Latent Semantic Indexing
- Bayes'sche Inferenz-Netzwerke
- ...

9.1 Das verallgemeinerte Vektorraummodell

Sowohl das Vektorraummodell als auch das BIR-Modell unterstellen die Unabhängigkeit der einzelnen Terme voneinander.

Für das Vektorraummodell wird diese Annahme normalerweise wie folgt gedeutet:

Sei $\vec{t_i}$ der Vektor, der mit dem Term t_i assoziiert wird, dann bedeutet Unabhängigkeit, dass die Menge der Vektoren $\{\vec{t_1}, \vec{t_2}, \ldots, \vec{t_t}\}$ linear unabhängig sind und eine Basis für den betrachteten Raum bilden.

Die Dimension dieses Raumes entspricht der Anzahl t der Terme.

Man kann dies auch restriktiver fassen, und fordern, dass $\vec{t_i} \bullet \vec{t_j} = 0$ für $i \neq j$ gelten muss.

($\vec{t_i} \bullet \vec{t_j} = 0$ stehe für das Skalarprodukt der beiden Vektoren.)

Dies bedeutet dann die „Orthogionalität" der Basisvektoren.

Man verwendet daher normalerweise $\vec{t_1} = (1; 0; 0; \ldots; 0)$, $\vec{t_2} = (0; 1; 0; \ldots; 0)$, ...

Im verallgemeinerten Vektorraummodell [WZRW89, WZW85] wird nun keine Orthogonalität der Basisvektoren mehr gefordert.

Vielmehr bestimmt man Basisvektoren, die die Abhängigkeiten zwischen den Termen wiederspiegeln sollen.

w_{dk} sei weiterhin das Gewicht das einem Paar aus Dokument D und Term t_k zugeordnet ist.

Wir definieren nun eine Menge mit 2^t Mintermen, die gegeben sind durch $m_1 = (0, 0, \ldots, 0)$, $m_2 = (1, 0, \ldots, 0) \ldots m_{2^t} = (1, 1, \ldots, 1)$.

$g_i(m_j)$ liefere zu dem Minterm j die i-te Komponente.

Damit zeigt der Minterm m_1 (mit $\forall i : g_i(m_1) = 0$) auf die Dokumente, die keinen der Terme enthalten. Minterm m_2 zeigt auf die Dokumente, die nur Term 1 enthalten und Minterm m_{2^t} zeigt auf die Dokumente, die alle Terme enthalten.

Nun bildet man mit 2^t paarweise orthogonalen Vektoren $\vec{m}_i$ einen neuen Raum:

$$\begin{aligned} \vec{m}_1 &= (1, 0, \ldots, 0, 0) \\ \vec{m}_2 &= (0, 1, \ldots, 0, 0) \\ &\vdots \\ \vec{m}_{2^t} &= \underbrace{(0, 0, \ldots, 0, 1)}_{2^t \text{ Komponenten}} \end{aligned}$$

Diese Menge von Vektoren wird dann als die orthogonale Basis des verallgemeinerten Vektorraummodells vewendet.

Die paarweise Orthogonalität im 2^t-dimensionalen Raum bedeutet dabei nicht die Unabhängigkeit unter den Indextermen.

Vielmehr spiegeln die einzelnen Dimensionen nun gerade die Bedeutung der Korrelationen zwischen den Termen wieder.

So ist der Vektor $\vec{m}_4$ mit dem Minterm $m_4 = (1, 1, 0, \ldots, 0)$ assoziiert, der ausschließlich auf die Dokumente verweist, die die Terme t_1 und t_2 enthalten.

Wenn es solche Dokumente gibt, sagen wir der Minterm m_4 ist *aktiv*. Dies bedeutet wiederum, dass es ein Indiz für eine Korrelation der Terme t_1 und t_2 gibt.

Um nun den 2^t-dimensionalen Indexvektor $\vec{t_i}$ für den Term t_i zu berechnen, werden die Vektoren der Minterme, in denen der Term t_i im Zustand 1 ist, aufaddiert und normiert:

$$\vec{t_i} = \frac{\sum_{\{r|g_i(m_r)=1\}} c_{i,r}\vec{m}_r}{\sqrt{\sum_{\{r|g_i(m_r)=1\}} c_{i,r}^2}}$$

$$c_{i,r} = \sum_{\{D|\forall l: g_l(\beta(D))=g_l(m_r)\}} w_{di} \quad \text{mit} \quad g_l(\beta(D)) = 1 \Leftrightarrow w_{dl} > 0$$

Beispiel:

$$\beta(D_1) = \begin{pmatrix} 0,4 \\ 0,9 \\ 0,0 \end{pmatrix} \quad \beta(D_2) = \begin{pmatrix} 0,7 \\ 0,0 \\ 0,7 \end{pmatrix} \quad \beta(D_3) = \begin{pmatrix} 1,0 \\ 0,0 \\ 0,0 \end{pmatrix} \quad \beta(D_4) = \begin{pmatrix} 0,0 \\ 1,0 \\ 0,0 \end{pmatrix} \quad \beta(D_5) = \begin{pmatrix} 0,7 \\ 0,7 \\ 0,0 \end{pmatrix}$$

Damit ergeben sich die folgenden acht Minterme: $m_1 = (0;0;0)$, $m_2 = (1;0;0)$, $m_3 = (0;1;0)$, $m_4 = (1;1;0)$, $m_5 = (0;0;1)$, $m_6 = (1;0;1)$, $m_7 = (0;1;1)$ und $m_8 = (1;1;1)$, von denen allerdings nur m_2, m_3, m_4 und m_6 aktiv sind.

Die $c_{i,r}$ werden nun gebildet, indem über alle Dokumente, deren „Profil“ dem Minterm m_r entspricht, die Gewichte w_{di} für den Term t_i addiert werden.

Für die $c_{i,r}$ erhalten wir:

	$c_{i,1}$	$c_{i,2}$	$c_{i,3}$	$c_{i,4}$	$c_{i,5}$	$c_{i,6}$	$c_{i,7}$	$c_{i,8}$
Minterm	$(0;0;0)$	$(1;0;0)$	$(0;1;0)$	$(1;1;0)$	$(0;0;1)$	$(1;0;1)$	$(0;1;1)$	$(1;1;1)$
$c_{1,r}$	0,0	1,0	0,0	1,1	0,0	0,7	0,0	0,0
$c_{2,r}$	0,0	0,0	1,0	1,6	0,0	0,0	0,0	0,0
$c_{3,r}$	0,0	0,0	0,0	0,0	0,0	0,7	0,0	0,0

Die Vektoren für die einzelnen Terme ergeben sich nun im Prinzip aus den Zeilen der obigen Tabelle, die noch normiert werden:

$$\vec{t_1} = \begin{pmatrix} 0 \\ 0,61 \\ 0 \\ 0,67 \\ 0 \\ 0,43 \\ 0 \\ 0 \end{pmatrix}, \vec{t_2} = \begin{pmatrix} 0 \\ 0 \\ 0,53 \\ 0,85 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ und } \vec{t_3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Man kann nun das Skalarprodukt $\vec{t_i} \bullet \vec{t_j}$ der Vektoren für die Terme t_i und t_j als Maß für die Korrelation der Terme betrachten.

Als Beschreibungsvektor für ein Dokument wird nun der Vektor $\mathcal{D}_* = \Sigma_{i=1}^{t} w_{di} \cdot \vec{t_i}$ verwendet.

Analog kann als Beschreibungsvektor für einen Anfragetext der Vektor $\mathcal{Q}_* = \Sigma_{i=1}^{t} w_{qi} \cdot \vec{t_i}$ verwendet werden.

Die Ähnlichkeit zwischen einem Dokument und einer Anfrage wird dann wieder über das Skalarprodukt berechnet.

Zu beachten ist dabei, dass die Dimensionen des 2^t-dimensionalen Raumes, die Mintermen entsprechen, die nicht aktiv sind, auch nicht berücksichtigt werden müssen.

Man hat es damit letztlich nicht mit einem 2^t-dimensionalen, sondern maximal mit einem N-dimensionalen Raum zu tun.

Da allgemein eher zweifelhaft ist, ob eine Berücksichtigung der Korrelationen zwischen Termen die Retrievalqualität erhöht, erscheint es letztlich fraglich, ob das verallgemeinerte Vektorraummodell bessere Ergebnisse verspricht, als das herkömmliche Vektorraummodell.

9.2 Latent Semantic Indexing

Zwei Effekte können bei der auf einzelnen Indextermen basierenden Suche zu Problemen führen:

- Zahlreiche nicht relevante Dokumente können im Ergebnis enthalten sein.
- Relevante Dokumente, die die in der Anfrage erwähnten Indexterme nicht enthalten, werden nicht gefunden.

Die Ursache hierfür liegt in der „eher zufälligen“ Verwendung von Begriffen in Dokumenten.

Möglichkeiten dem entgegenzuwirken:

- Die Erweiterung der Anfrage mit Hilfe eines Thesaurus („related terms“ werden zusätzlich in die Anfrage aufgenommen).
- Verwendung des verallgemeinerten Vektorraummodells
- Latent Semantic Indexing
- ...

Die Ideen in einem Text hängen stärker mit den im Text beschriebenen Konzepten als mit den verwendeten Indextermen zusammen!

$\Rightarrow$ die Bestimmung der Retrievalgewichte sollte sich auf diese Konzepte beziehen!

Die Idee des *Latent Semantic Indexing* [DDF+90] ist nun wie folgt:

Die Beschreibungsvektoren für Dokumente und Anfragen werden in einen Raum geringerer Dimensionalität transformiert, dessen Dimensionen nicht den Indextermen, sondern den Konzepten entsprechen!

Im weiteren sei nun $\vec{M} = (M_{ij})$ die Term×Dokument-Matrix mit t Zeilen und N Spalten.

Diese Matrix kann z.B. gebildet werden, indem man die nach der $tf \cdot idf$-Formel berechneten t-dimensionalen Vektoren der N Dokumente als Spalten der Matrix verwendet.

Diese Matrix $\vec{M}$ wird nun mit Hilfe der Eigenwert-Dekomposition wie folgt aufgebrochen:

$$\vec{M} = \vec{K}\vec{S}\vec{D}^t$$

Die Teilmatrizen haben dabei folgende Interpretation:

- $\vec{K}$ ist die Matrix der Eigenvektoren, die sich für die Term×Term-Korrelationsmatrix ergeben.
  Diese Term×Term-Korrelationsmatrix wiederum ist eine $t \times t$-Matrix die durch $\vec{M}\vec{M}^t$ definiert ist und die Korrelation des Auftretens der Terme in den Dokumenten beschreibt.
- $\vec{D}^t$ ist die Matrix der Eigenvektoren, die sich für die transponierte Dokument×Dokument-Korrelationsmatrix ergeben.
  Diese Dokument×Dokument-Korrelationsmatrix ist eine $N \times N$-Matrix die durch $\vec{M}^t\vec{M}$ definiert ist, und die Korrelation zwischen den einzelnen Dokumenten beschreibt.
- $\vec{S}$ ist schließlich eine $r \times r$-Diagonalmatrix (mit $r = \min(t, N) =$ dem Rang der Matrix $\vec{M}$).

Nun wird die Matrix $\vec{S}$ reduziert, indem nur noch die s größten Eigenwerte von $\vec{S}$ und in der Folge auch nur noch die entsprechenden Spalten von $\vec{K}$ und $\vec{D}^t$ betrachtet werden.

Die sich ergebende Matrix $\vec{M}_s$ ist die Matrix vom Rang s, die im Sinne der Abweichungsquadrate am ähnlichsten zu $\vec{M}$ ist.

Es gilt $\vec{M}_s = \vec{K}_s\vec{S}_s\vec{D}_s^t$, wobei s die reduzierte Dimensionalität den Konzeptraumes ist.

s sollte dabei so gewählt werden, dass alle Konzepte erhalten bleiben und gleichzeitig die für das Retrieval unerheblichen (und eher störenden) Repräsentationsdetails eliminiert werden.

Bilden wir nun die Matrix $\vec{M}_s^t\vec{M}_s$ so erhalten wir die Dokument×Dokument-Korrelationsmatrix im Hinblick auf die betrachteten Konzepte.

Die Komponente (i, j) dieser Matrix gibt dabei die Korrelation zwischen den entsprechenden Dokumenten an.

Betrachten wir nun eine Anfrage als neues „Pseudodokument“ D_0, so erhalten wir in der ersten Zeile von $\vec{M}_s^t \vec{M}_s$ die Korrelationen der Dokumente zu dieser Anfrage, die uns direkt das entsprechende Ranking liefern.

Ist nun $s \ll t$ und $s \ll N$, so ergibt sich ein relativ effizientes Indexierungsmodell, in dem das „Rauschen“, dass durch die auf Indextermen basierende Repräsentation entsteht, weitgehend unterdrückt wird.

Ob sich damit allerdings gegenüber einfacheren Retrievalmodellen deutliche Steigerungen in der Retrievalqualität erzielen lassen, ist letztlich noch offen.

9.3 Bayes'sche Inferenznetze

Bayes'sches Inferenznetz [Pea88]:

Gerichteter azyklischer Graph, dessen Knoten Zufallsvariablen (Aussagen) und dessen Kanten Abhängigkeiten zwischen diesen Variablen (Aussagen) darstellen.

Die Knoten können Wahrscheinlichkeitswerte zwischen 0 und 1 annehmen.

Wahrscheinlichkeiten dienen als Ausgangswerte zur Berechnung neuer Wahrscheinlichkeiten in den Knoten, auf die die Kanten zeigen.

Dazu besitzt jeder Knoten eine Funktion oder Tabelle, die aus der Wahrscheinlichkeit der Knoten, die auf ihn zeigen, eine neue Wahrscheinlichkeit berechnet.

Im einfachsten Fall, wenn es zu einem Knoten nur einen anderen Knoten gibt, der auf ihn zeigt, kann dies eine einfache bedingte Wahrscheinlichkeit sein.

Anwendung im Information Retrieval (nach Turtle und Croft [TC90, TC91]):

Man verwendet ein zweigeteiltes Inferenznetz das in Schichten organisiert ist.

Der eine Teil, das Document Network (Dokumentennetz) besteht aus drei Schichten von Knoten:

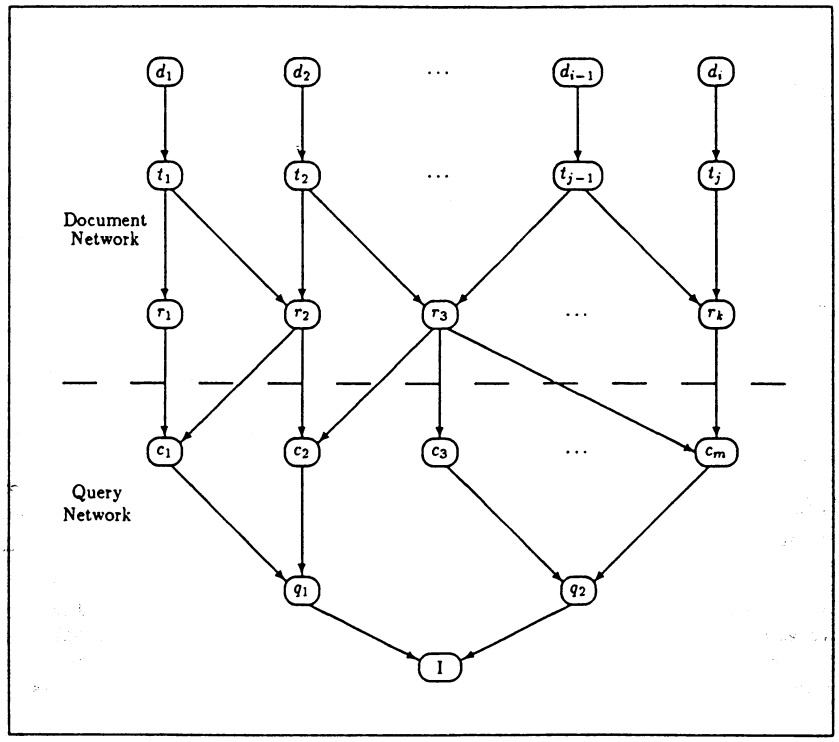
1. der Dokumentenschicht,
2. der Textrepräsentationsschicht und
3. der Konzeptrepräsentationsschicht.

Kanten existieren nur zwischen diesen Schichten

- von Dokumentknoten zu Textrepräsentationsknoten und
- von Textrepräsentationsknoten zu Konzeptrepräsentationsknoten.

Dieser Teil des Netzes wird nur von den Dokumenten bestimmt.

Inferenznetz für das Information Retrieval nach Turtle und Croft [TC90]:



Der zweite Teil des Netzes, das Query Network (Anfragenetz) kann ebenfalls aus mehreren Schichten bestehen:

- die Knoten der ersten Schicht korrespondieren mit den Konzepten, die den Informationsbedarf ausdrücken, und sind folglich mit der Konzeptrepräsentationsschicht des Document Network verbunden
- die letzte Schicht besteht nur aus einem Knoten, der mit die Wahrscheinlichkeit, dass ein Informationsbedarf befriedigt werden kann, entspricht
- durch Verwendung der Zwischenknoten kann ein Informationsbedarf mit Hilfe mehrerer Anfragen ausgedrückt werden.

Dieser Teil des Netzes ist nur von der jeweiligen Anfrage abhängig und nicht von den Dokumenten.

Wie können wir nun in einem solchen Netz die Inferenz berechnen?

1. Möglichkeit:

- Für jedes Dokument jeweils den entsprechenden Knoten auf 1 setzen und alle anderen Knoten auf 0.
- Die Wahrscheinlichkeit pflanzt sich dann durch das Netz fort und man kann
- im letzten Knoten die Relevanz ablesen.

→ Verschiedene Pfade durch das Netz repräsentieren verschiedene Aspekte, die zur Beurteilung der Wichtigkeit eines Dokumentes beitragen.

2. Möglichkeit:

- Die optimale Ergebnismenge bestimmen, indem man die Teilmenge der Dokumente ermittelt, die gemeinsam den höchsten Nutzen erbringen. (Voraussetzung: „Opportunitätskosten“ für die Betrachtung eines nicht relevanten Dokuments)
- Dies ist aber im allgemeinen Fall zu aufwendig.

Die Dokumentenschicht:

In der ersten Schicht des Netzes werden die verschiedenen Dokumente der Sammlung repräsentiert.

Dokumente in eine Rangfolge bringen:

Werte der Dokumentknoten einzeln auf Eins setzen und Wahrscheinlichkeitswert der Inferenz berechnen.

Die Textrepräsentationsschicht

Knoten stehen für verschiedene Textteile, -typen oder -sichten.

Dabei kann ein Dokument mehrere Textrepräsentationsknoten besitzen.

Verschiedene Abschnitte eines Dokuments, Titel, Abstrakt, Kapitel, Teile, Bilder, Audio- oder Videodaten, Kommentare, Klassifikationen, Indexterme, Einschätzungen durch Gutachter.

Die Konzeptrepräsentationsschicht

Die Knoten dieser Schicht repräsentieren abstrakte Konzepte.

Ihre Aktivierungen werden aus denen der Textrepräsentationsknoten berechnet.

Sie stellen die Schnittstelle zum Anfragenetz dar.

Drei Schichten des Dokumentennetzes:

→ zunehmende Abstrahierung der Dokumentinhalte von der jeweiligen Darstellungsform.

Bestimmung der Übergangsfunktionen:

→ sollte einfach genug sein, um für große Dokumentensammlungen berechnet werden zu können.

Aus dem theoretischen Modell leiten Turtle und Croft [TC91] ein weiter vereinfachtes Retrievalsystem INQUERY ab.

Dazu

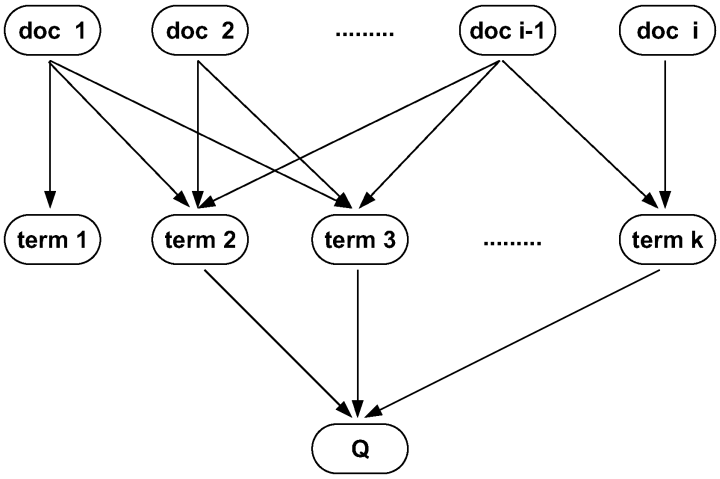
- verzichten sie auf die Textrepräsentationsschicht,
- leiten die Konzepte unmittelbar aus den Termen, die in den Dokumenten vorkommen, her,
- verzichten auf die Anfrageschicht im Anfragenetz
- verwenden anstelle der Wahrscheinlichkeiten allgemeinere Werte, die aus einem IDF-Maß und anderen Häufigkeitsmaßen der Terme gewonnen werden, und
- ersetzen die individuellen Funktionen in den Knoten durch eine allgemeine Funktion. Dabei gehen sie von einer Unabhängigkeitsannahme aus.

→ als invertierte Liste mit Wahrscheinlichkeitswerten implementiert

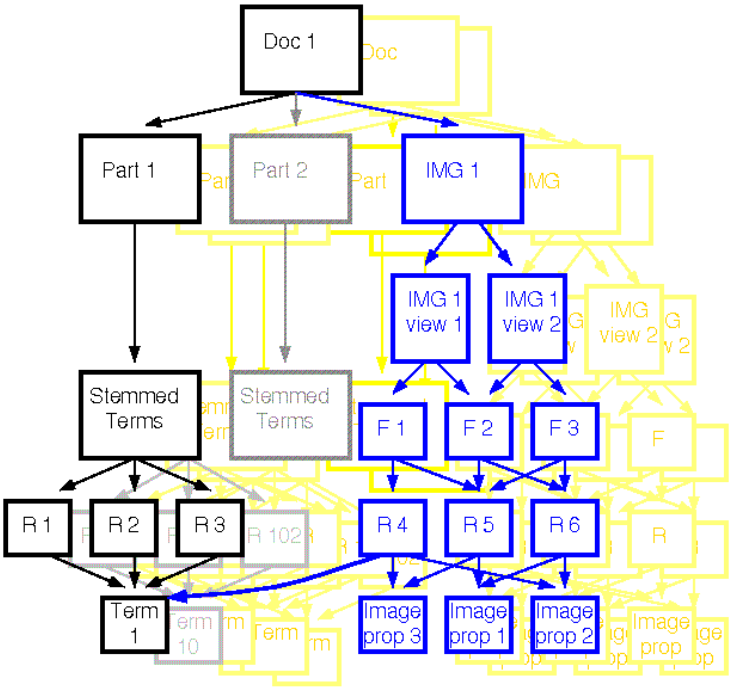
→ in der Lage, sehr grosse Dokumentsammlungen zu verwalten.

→ reduziert sich im wesentlichen auf ein gewichtetes Vektorraummodell.

Inferenznetz, wie es zur Implementierung von INQUERY verwendt wurde

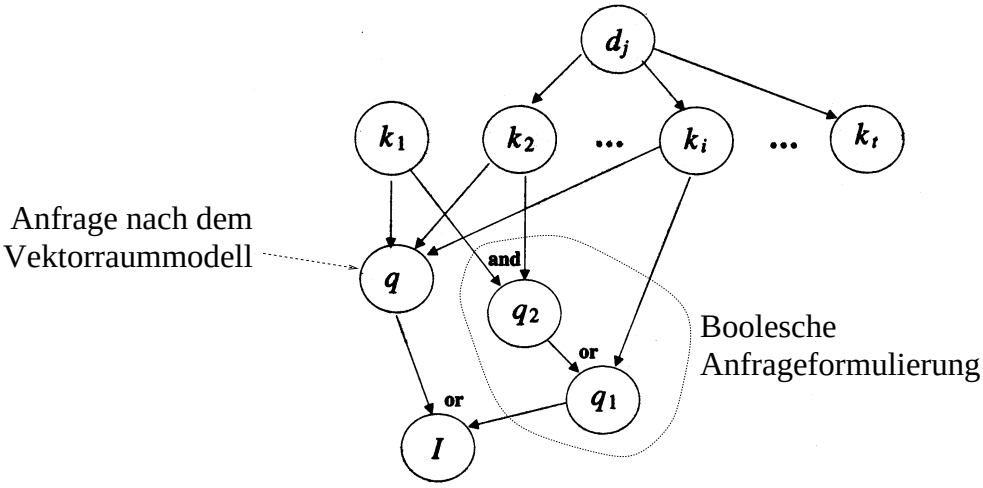


Inferrenznetz des MAGIC Systems [Fer99]
(MAGIC = Multimedia-based Automatic Generation of Indexes and Clusters)



Zusammenschalten mehrerer Ansätze im Netz

z.B. Boolesches Retrieval und Vektorraummodell [BYRN99]:



Die Übergänge müssen dann entsprechend gewählt werden.

Damit kann ein Informationswunsch alternativ und/oder ergänzend in verschiedenen Formen repräsentiert werden.

Kapitel 10

Information Retrieval ↔ Datenbanken

- Integration von Vagheit in Datenbanken [Fuh90, FR97]
- Integration von Textdokumenten und entsprechenden Zugriffsoperationen in Datenbanken
- Verwaltung von Dokumenten in strukturierter Form in objektorientierten Datenbanken
- Integration auf der Ebene der Anfragesprache
- Integration auf der Ebene der Zugriffsstrukturen
- ...

10.1 Der Bedarf zur Kombination

Stärken:

- *relationale Datenbanken:*
  - effiziente Verarbeitung großer Datenmengen
  - Suche mit klaren Bedingungen
- *objektorientierte Datenbanken:*
  - Bearbeitung strukturierter Daten
  - Definition neuer Datentypen
- *Information Retrieval:*
  - Verwaltung von Textdokumenten
  - Ranking im Ergebnis
  - potentielle Erweiterbarkeit auf Multimedia-Dokumente

1. Beispiel: Auswahl in einer Datenbank

Das Problem:

In einer Datenbank mit Gebrauchtwagen soll ein passendes Exemplar gesucht werden!

Gebrauchtwagen

G-NR	MODELL	KLIMA	BAUJAHR	KW	KILOMETER	PREIS
132	601	ja	1994	45	19000	27000
472	601	ja	1993	80	10000	17000
173	601	ja	1995	45	23000	19000
⋮	⋮	⋮	⋮	⋮	⋮	⋮

- Gesucht wird: ein 601
  - mit weniger als 20000 km
  - nach Baujahr 1993
  - mit mindestens 45 KW
  - möglichst günstig

- die Anfrage in SQL:

```
SELECT G-NR FROM Gebrauchtwagen WHERE KILOMETER < 20000
  AND BAUJAHR > 1993
  AND KW >= 45
ORDER PREIS ASC
```

Gebrauchtwagen

G-NR	MODELL	KLIMA	BAUJAHR	KW	KILOMETER	PREIS
132	601	ja	1994	45	19000	27000
472	601	ja	1993	80	10000	17000
173	601	ja	1995	45	23000	19000
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem: die Bedingungen sind eigentlich vage!

2. Beispiel: Vertragsanalyse

Das Problem: *Umfangreiche Vertragswerke sollen auf die in Ihnen enthaltenen Vertragsrisiken untersucht werden!*

Lösungsansatz: *Zu allen risikorelevanten Bereichen werden Anfragen gestellt, die Vertragsabschnitte finden, die sich potentiell mit diesen beschäftigen.*

Wie werden die Anfragen formuliert?

- es gibt eine Menge von Musterverträgen
- in den Musterverträgen werden manuell Passagen ermittelt, die bestimmte risikorelevante Aspekte regeln
- für die zu einem Aspekt gefundenen Passagen wird der Zentroid über alle Beschreibungsvektoren gebildet
- dieser Zentroid gilt als Anfrage und kann ständig weiterentwickelt werden

Basis zur Vertragsverwaltung:

- ein objektorientiertes Datenbanksystem zur Verwaltung der strukturierten Dokumente

3. Beispiel: Wiederverwendbare Komponenten

Das Szenario:

- Es existiere ein landesweiter Pool
  - in den Dozenten ihre Vorlesungsunterlagen einstellen
  - auf den Dozenten zum Zwecke der Wiederverwendung zugreifen können
- In diesem Pool kann man suchen nach
  - kompletten Vorlesungen
  - einzelnen Kapiteln
  - Grafiken, Animationen
  - Übungsaufgaben, ...

Ein Anfragesystem für ein solches System muß

- die Dokumente strukturiert verwalten
- Faktenanfragen erlauben
- Texte und andere Medien adressieren können

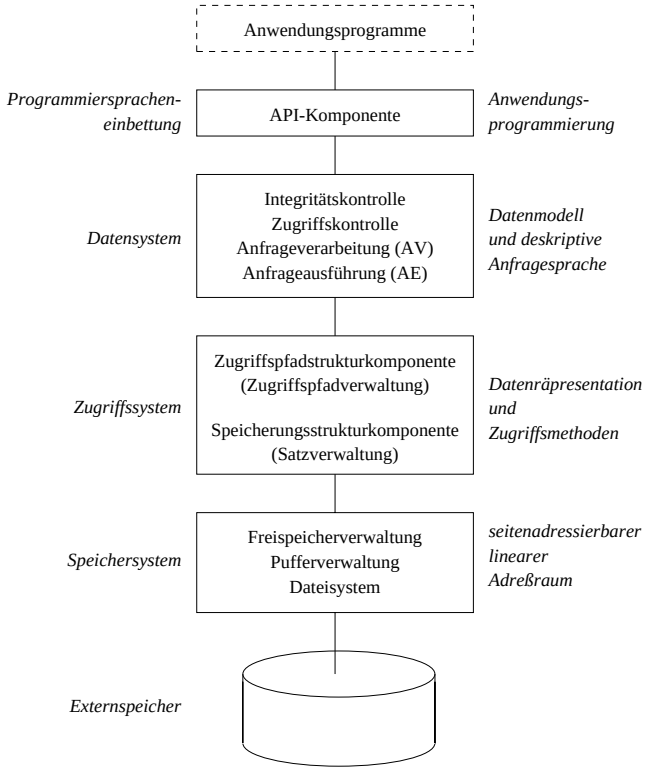
Beispielanfrage:

Suche eine Grafik in einem der Formate ..., die im Kontext der Erzeugung von binären Bäumen auftritt und mindestens 10 Kanten enthält.

Das Ergebnis sollte ein Ranking sein.

Skizzen könnten als Anfrage ebenfalls sinnvoll sein.

Komponentensicht eines Datenbank-Managementsystems [Mit95]



10.2 POQL – Eine Anfragesprache mit IR-Funktionalitäten

Gliederung [Hen96]:

1. Motivation
2. Addressing a Document
3. Pattern Matching
4. Term-Based Document Retrieval
5. Conclusion

Motivation

- Während der Softwareentwicklung werden zahlreiche *„Dokumente“* erstellt:
  - Textdokumente (wie z.B. Projektvorschläge)
  - Diagramme (wie z.B. OOA-Diagramme)
  - Testberichte
  - Programmcode
  - Designdokumente (wie z.B. Moduldefinitionen)
  - ...
- Dokumente enthalten wichtige textuelle Bestandteile
- Dokumente werden in einem Repository verwaltet

  ⇒ **Information Retrieval für Software-Entwicklungsumgebungen**
- Um wiederverwendbare „Dokumente“ zu suchen.
- Um redundante „Dokumente“ zu suchen.
- Um verwandte „Dokumente“ zu suchen.

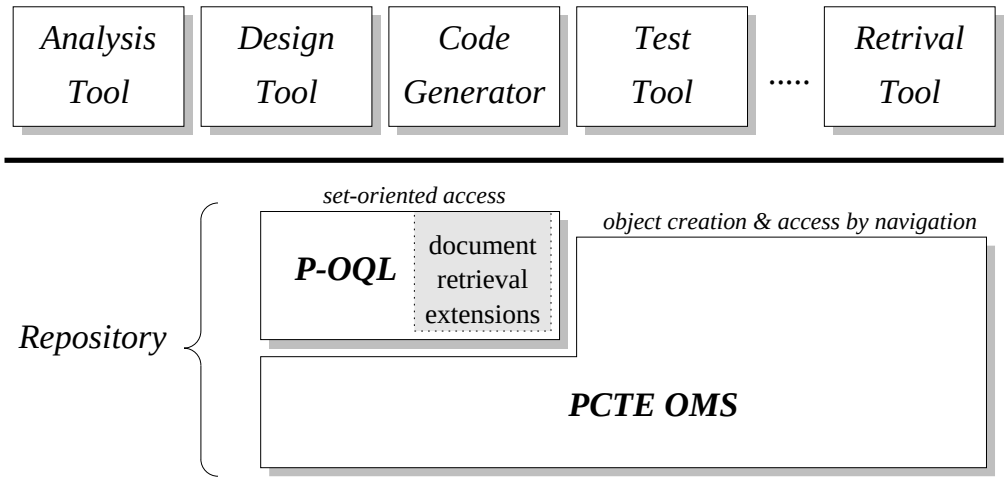
Die Umgebung für unsere Überlegungen:

PCTE: (Portable Common Tool Environment) the ISO/ECMA standard for a public tool interface (framework) for system development environments

PCTE provides tool developers with

- communication services
- process management services
- *object management services*

P-OQL: (PCTE Object Query Language)
an OQL-like query language for the object management system of PCTE



The data model of PCTE

- structurally object-oriented (extended ER-Model)

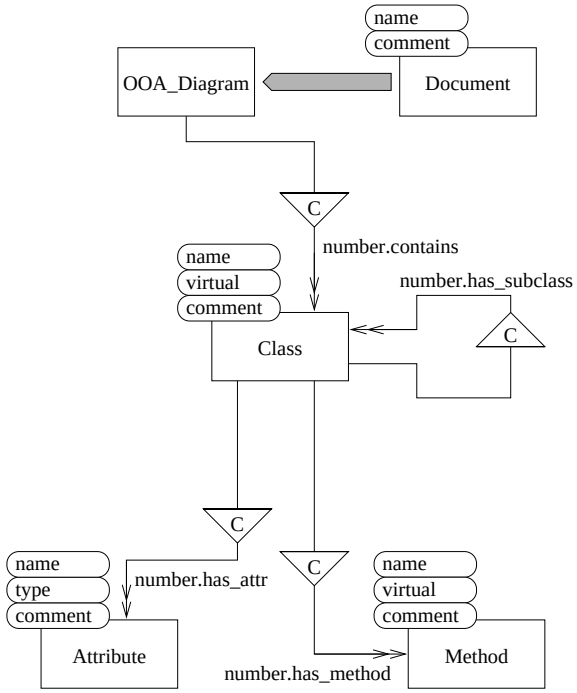
Building blocks:

- *object type*
  - name
  - applied attributes
  - allowed outgoing link types
  - supertypes
- *attribute*
  - name
  - value type (boolean, **string**, integer, real, time, enumeration types)

- *link type (≈ relationship type)*
  - name
  - key and non-key attributes
  - allowed destination object types
  - category

category	**property**			
	composition	existence	referential integrity	relevance to origin
composition:	+	+	+	+
existence:	−	+	+	+
reference:	−	−	+	+
implicit:	−	−	+	−
designation:	−	−	−	+

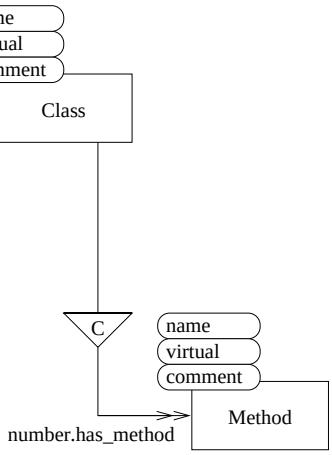
Example schema (OOA-diagrams)



Example query in P-OQL

Get the name and the name of all methods for virtual classes:

```
select name, _.has_method/->name
   from Class
   where virtual = true
```



2. Addressing a document

Problem: What is a "document" in our context?

Examples

- *a class definition*
  together with the associated attribute and method definitions
- *an OOA-diagram*
  together with all objects which can be reached via *composition* links
- *a Document*
  together with all objects which can be reached via *composition* links

⇒ A flexible way to define what should be addressed
as a document in a given query is needed

Solution: The objects and links forming a *document* are addressed
by **regular path expressions**

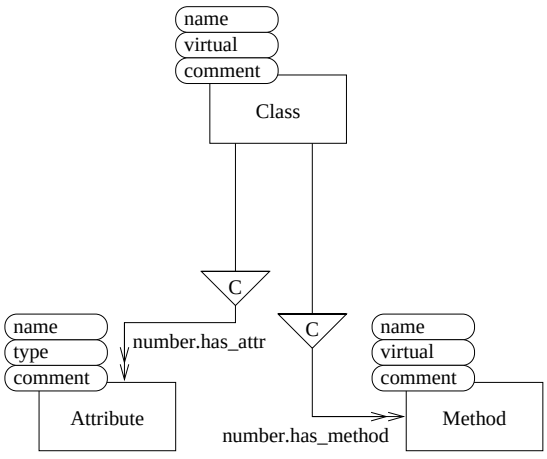
regular path expressions

- use the anchor object of the document as their starting point
- define the paths leading to the *"components"* of the document
- yield a (structured) collection of objects, links or string attributes

Example 1 (based on link types):

Search for the *class definitions*:

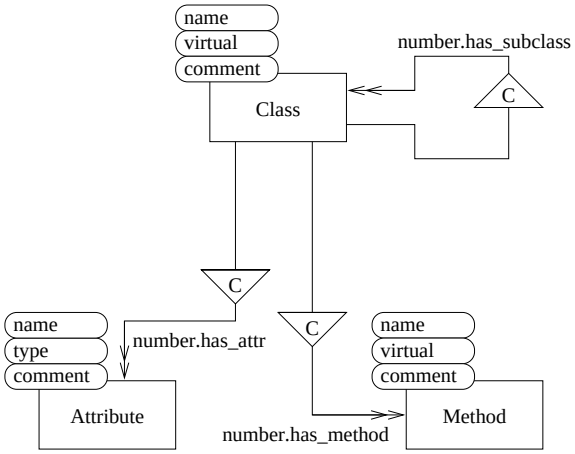
```
select (., _.has_attr/->., _.has_method/->.)
   from Class
```



Example 2 (based on categories):

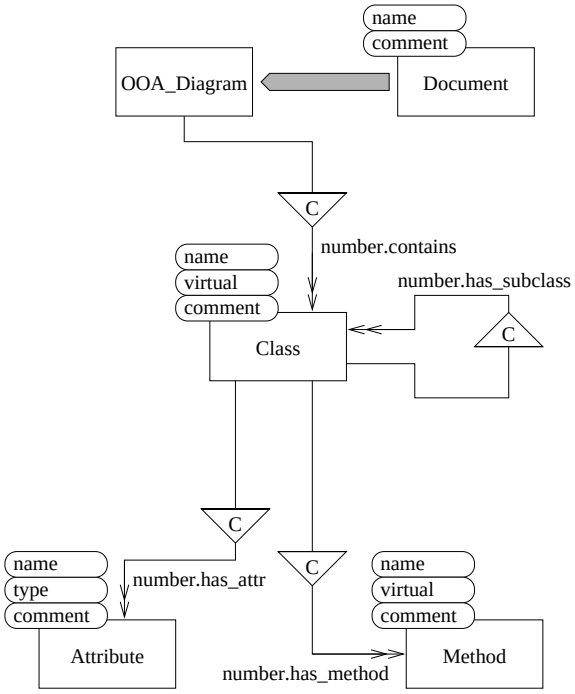
Search for the *class definitions*:

```
select (., {c shield Class}/->.)
   from Class
```



Example 3:

Search for all *Documents*: `select [{c}]*/->. from Document^`



3. Pattern matching

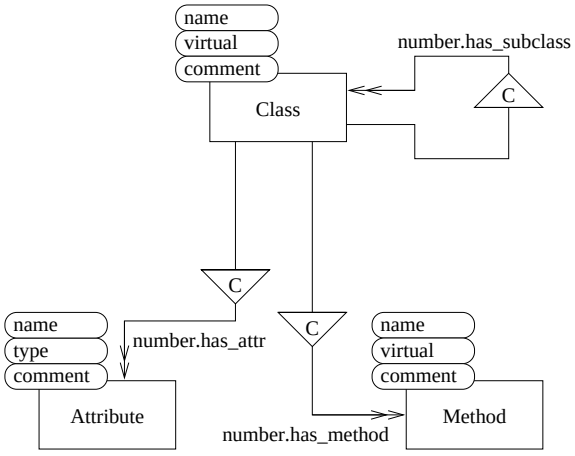
similar to `grep` command in UNIX

- `grep` operator in P-OQL
- syntax: <*regular expr.*> `grep` <*document definition*>
- yields a set with an entry for each line in the *"document"* containing a substring matching the regular expression
- each element in this set contains information about the location of the matching substring:
  - object or link reference
  - attribute name
  - line number
  - position and length

Example 4:

Search for the *class definitions* containing a substring matching *"[Bb]illing"* in their *comment* Attribute:

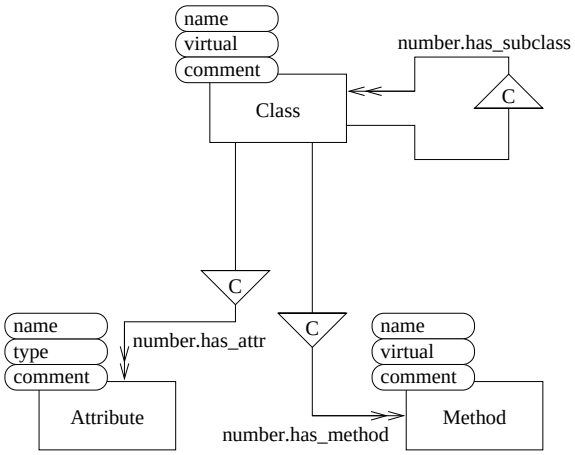
```
select ., ("[Bb]illing" grep comment)
   from Class
   where 0 < count ("[Bb]illing" grep comment)
```



Example 5:

Search for the *class definitions* containing a substring matching *"[Bb]illing"*:

```
select ., ("[Bb]illing" grep (., {c shield Class}/->.))
   from Class
   where 0 < count ("[Bb]illing" grep (., {c shield Class}/->.))
```



4. Term-based document retrieval

We introduced operators to calculate document representation vectors.

The unary Operator `D_vector` calculates a document representation vector

$$\mathcal{D} = (w_{d1}, w_{d2}, \ldots, w_{dt})$$

according to

$$w_{dk} = \frac{tf_{dk} \cdot \log \frac{N}{n_k}}{\sqrt{\sum_{i=1}^{t} \left(tf_{di} \cdot \log \frac{N}{n_i} \right)^2}}$$

syntax: `D_vector` <*document definition*>

The unary Operator `Q_vector` calculates a representation vector for a query text

$$\mathcal{Q} = (w_{q1}, w_{q2}, \ldots, w_{qt})$$

according to

$$w_{qk} = \begin{cases} \left(0.5 + \frac{0.5 \cdot tf_{qk}}{\max\limits_{1 \le i \le t} tf_{qi}}\right) \cdot \log \frac{N}{n_k} & \text{if} \quad tf_{qk} > 0 \\ 0 & \text{if} \quad tf_{qk} = 0 \end{cases}$$

syntax: `Q_vector` <*document definition*>

The binary Operator `sim` calculates the similarity for two vectors according to

$$\text{similarity}(\mathcal{Q}, \mathcal{D}) = \sum_{k=1}^{t} w_{qk} \cdot w_{dk}$$

syntax: <*vector definition*> `sim` <*vector definition*>

Examples:

Search for all documents concerned with OMS and OQL:

```
head[25]
  sort- (
    select (Q_vector "OMS and OQL" sim D_vector [{c}]*/->.),
           .
       from Document^ )
```

Search for the Dokumente most similar to the OOA-diagram named *"Accounting"*:

```
head[25]
  sort- (
    select (Q_vector Q:[{c}]*/->. sim D_vector D:[{c}]*/->.),
           D:.
       from Q in OOA_Diagram,
            D in Document^
       where Q:name = "Accounting"
         and Q:. != D:. )
```


The described operators can also be used to find e.g. pairs of similar Dokumente.

Example:

Find similar class definitions which are not subclasses of each other:

```
head[25]
 sort- (
  select (D_vector A:(., {c shield Class}/->.)
          sim
          D_vector B:(., {c shield Class}/->.)),
         A:.,
         B:.
     from A in Class,
          B in Class
     where A:. != B:.
       and A:. !in B:[_.has_subclass]+/->.
       and B:. !in A:[_.has_subclass]+/->. )
```


Problem: Vocabulary and Collection Wide Information (N, n_k)

Each query defines its own document set

$\Rightarrow$ open problems

- Which vocabulary should be used?
- How to calculate the collection wide information (CWI)

Different solutions are possible:

- use a vocabulary and CWI calculated for a *"typical"* set of Dokumente
- use a global vocabulary and calculate the CWI for each query
- calculate the vocabulary and the CWI for each query
- ...

5. Conclusion

Document Retrieval Facilities for Repositories

Topics addressed:

- document definition
- pattern matching
- search for similar Dokumente

Related Topics:

- an access structure for combined queries (→ CIKM '96)
- a user interface hiding the syntax of P-OQL

Future Research:

- recall-precision experiments with different strategies for the vocabulary and the CWI
- incorporation of relevance feedback
- ...

Kapitel 11

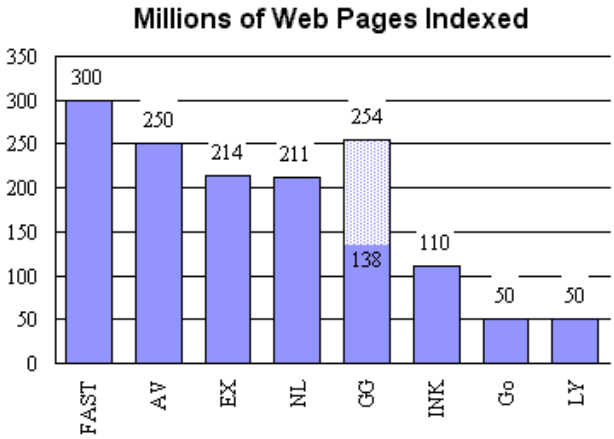
Suchmaschinen im Internet

- Architektur von Suchmaschinen
- Benutzungsschnittstellen
- Ranking und Zugriffsstrukturen
- Kataloge
- Anfragesprachen für das Web

Herausforderungen für IR im Internet:

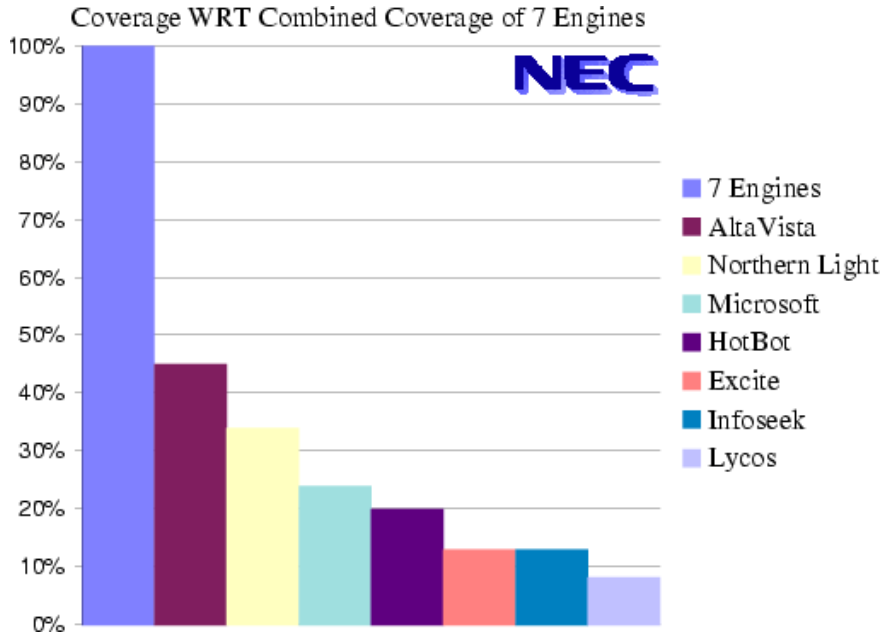
- die Daten sind weltweit verteilt
- der Datenbestand ist sehr unbeständig
  - es kommen ständig neue „Seiten“ hinzu
  - bestehende Seiten ändern sich
  - bestehende Seiten sind nicht mehr verfügbar
- die Daten haben einen enormen Umfang
- die Daten sind sehr heterogen und im allgemeinen wenig strukturiert
- die Qualität der Daten ist sowohl inhaltlich als auch sprachlich (Tippfehler, OCR-Fehler, ...) problematisch
- bei den Daten handelt es sich zum Teil um Multimedia-Daten

Umfang der Suchmaschinen
(nach `http://www.searchenginewatch.com/reports/sizes.html`):



Legende: FAST=FAST, AV=AltaVista, EX=Excite, NL=Northern Light,
GG=Google, INK=Inktomi, Go=Go (Infoseek), LY=Lycos.

Here is the estimated coverage of each engine compared with the combined coverage of the 7 engines used in the study. These estimates are averaged over 1025 queries performed in September 1998.



11.1 Architektur von Suchmaschinen

Suchmaschinen müssen zwei Bereiche abdecken:

- Aufnahme von Seiten in den Datenbestand und Aktualisierung
- Bearbeitung von Anfragen

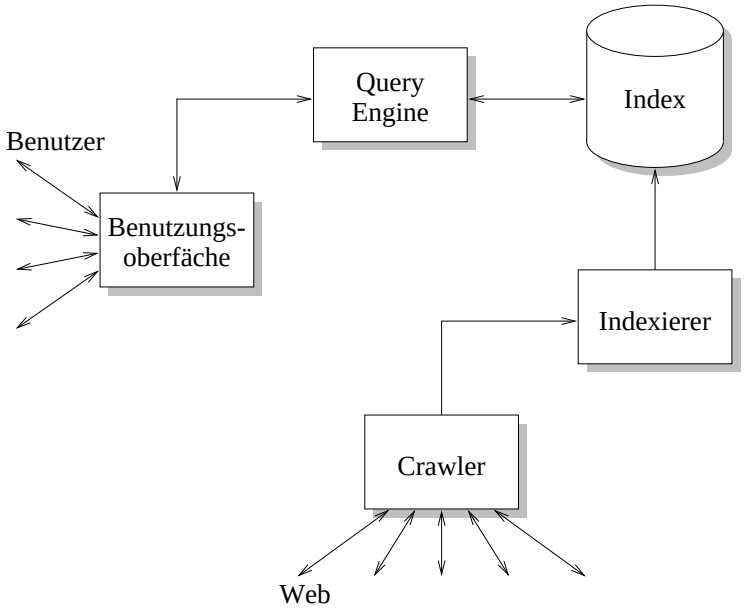
Die erste Aufgabe übernehmen typischerweise sogenannte Crawler [auch Robots, Walker, Scooter (Alta Vista), ArchitextSpider (Excite), Slurp the Web Hound (HotBot), oder Spidey (Web Crawler)].

Diese durchsuchen von Rechnern des Suchmaschinenbetreibers aus das Netz.

Sie laufen dabei (entgegen ihrem Namen) immer zentral auf den Rechnern des Betreibers der Suchmaschine und durchsuchen von dort das Netz.

Sie liefern die dabei gefundenen Dokumente an den Indexierer, der sie in den Index aufnimmt.

Typische zentralisierte Architektur:



Wie gehen die Crawler vor?

Einfachste Vorgehensweise:

Man startet mit einem Satz von URLs und extrahiert von dort andere URLs, die rekursiv mit einer Breiten- oder Tiefensuche durchlaufen werden.

Die meisten Suchmaschinen erlauben eine eigene Seite zur Menge der Start-URLs hinzuzufügen (z.B. *Add a URL* bei Altavista).

Eine Alternative ist, von sehr häufig zugegriffenen URLs zu starten, weil man annimmt, dass man von diesen Seiten über recht kurze Pfade zu anderen wichtigen Seiten kommt.

Die Information, welche Seiten am häufigsten zugegriffen werden, werden dabei z.B. von `www.directhit.com` bereitgestellt.

Probleme ergeben sich bei allen Ansätzen, wenn mehrere Crawler parallel arbeiten.

In diesem Fall kann man das Web z.B. nach den Endungen aufteilen oder nach anderen Kriterien.

- zwischen des Crawlern muss dann ein Austausch noch zu betrachtender Adressen erfolgen
- z.B. Verwaltung der noch zu betrachtenden URLs in einer Warteschlange je Crawler, in die alle Crawler Einträge machen können

Die Architektur führt natürlich dazu, dass die Aktualisierung der Indexe eine gewisse Zeit braucht (Angaben nach [Kar99]):

- So schaffen die Crawler von Fireball 500.000 URLs in 24 Stunden.
- Die Alta Vista Indizierungssoftware bewältigt pro Stunde ein Gigabyte Text. Der Gesamtindex beträgt 40 Gigabyte. 3 Millionen Seiten werden täglich gescannt.
- Nach [BYRN99] schaffen die schnellsten Crawler bis zu 10 Mio. Seiten pro Tag. Bei 300 Mio. indexierten Seiten vergehen so 30 Tage bis der Index einmal überarbeitet ist!

Dies führt dazu, dass die Suchmaschine einen Stand des Web „kennt“, den es nie gegeben hat (→ *globaler Zustand* in verteilten Systemen)

Die Folge ist, dass die Suchmaschine immer mit veralteten Indexierungsdaten arbeitet.

→ Seiten, auf die verwiesen wird, müssen nicht mehr existieren.

→ Der Inhalt von Seiten, auf die verwiesen wird, kann sich geändert haben.

Typischerweise wird dabei der Index ca. einmal pro Monat komplett aktualisiert.

Wie entscheidet ein Crawler welche Seite er als nächstes betrachten soll?

Ausgehend von den Start-URLs wird unterschiedlich tief gesucht:

- Maschinen wie Alta Vista und Excite verfolgen Links über drei und mehr Linkebenen.
- Andere Maschinen betrachten nur die Links der ersten Seite und ignorieren alle weiterführenden Links der zweiten Ebene, sofern es keine externen Links sind.

Es gibt nun Suchmaschinen, die sich bei der Indexierung von Seiten deren Änderungsdatum merken.

Stellen sie beim Aktualisieren des Verweises auf eine Seite fest, dass die Seite häufig geändert wird, betrachten sie diese Seite ggf. öfter beim Aktualisieren.

Berücksichtigt werden kann zur Bestimmung der Wichtigkeit einer Seite ferner

- die Zahl der Verweise auf eine Seite (und die Wichtigkeit der Seiten von denen aus verwiesen wird)
- die Zugriffshäufigkeit auf die Seite
- (Dabei werden „junge“ Seiten natürlich benachteiligt. Man vesucht dies zu kompensieren, indem Verweise und Zugriffe auf junge Seiten höher gewichtet werden.)

Zusätzlich kann man für einen Web-Server und für einzelne Dateien Steuerinformationen für Crawler angeben, die die meisten auch beachten:

- Mit den META-Tags *description* und *keywords* kann man zu einem Dokument gezielt Informationen für die Suchmaschinen angeben.
- Mit dem META-Tag *robots* kann man den Zugriff der Crawler auf eine Seite steuern.
  Bsp.: `<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">`
- Mit der `robots.txt`-Datei kann durch den *Robot Exclusion Standard* eine genaue Kontrolle erfolgen, welche Verknüpfungen einer Website von Crawlern betrachtet werden sollen.
  So verbietet z.B. die Datei

```
User-agent: * # directed to all robots
Disallow: /
```

  Crawlern die Betrachtung der ganzen Website.

Ein verteilter Ansatz zur Indexierung

Problem des bisher beschriebenen zentralen Ansatzes für Crawler:

- Crawler erzeugen sehr viel Verkehr im Netz (ca. 7 % des Verkehrs)
- Seiten werden von vielen verschiedenen Crawlern betrachtet

⇒ Wieso liefert nicht eine dezentrale Indexierungsinstanz Indexierungsinformationen zu bestimmten Seiten an die Suchmaschinen?

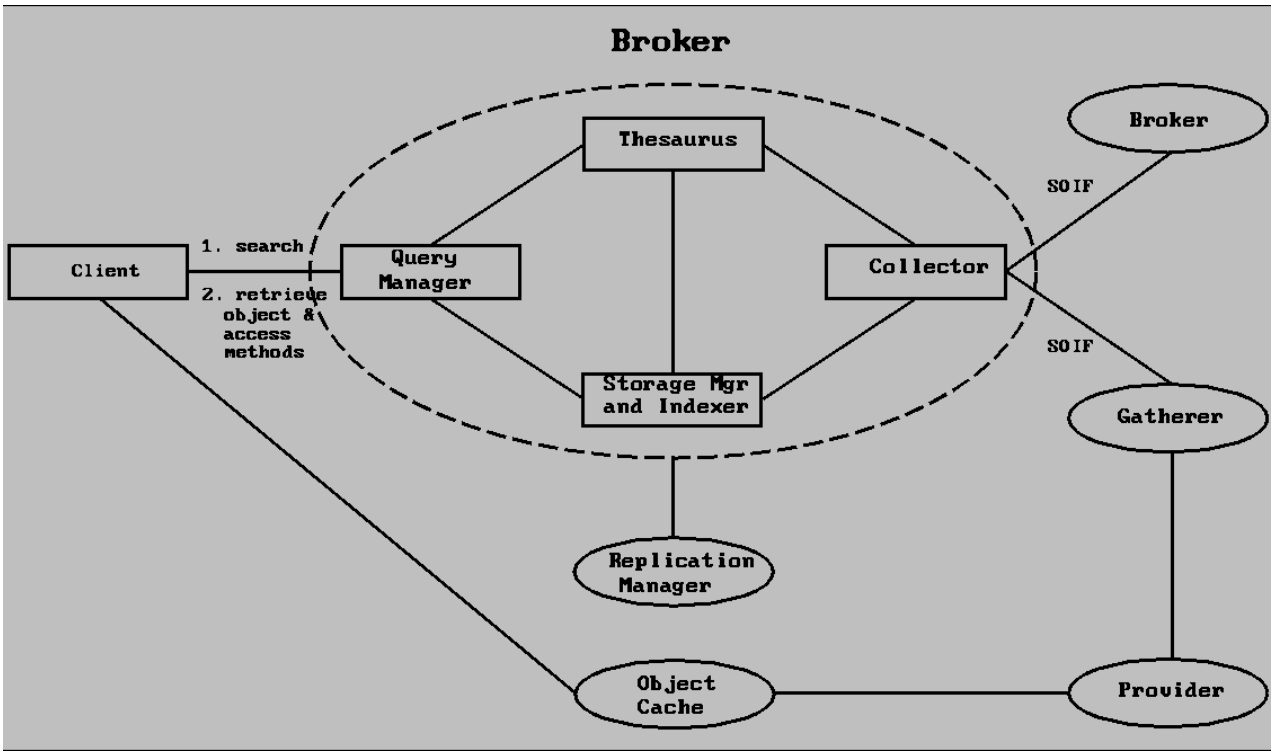
Ein Ansatz hierzu: **Harvest** (übernommen aus http://www11.informatik.tu-muenchen.de/lehre/seminarSS96fi/broker.html)

- Entwicklung an der Universität von Colorado mit Unterstützung der „Advanced Research Projects Agency“ und in Zusammenarbeit mit anderen Hochschulen.
- Modular konzipiertes Informationssystem, welches einen Gatherer, einen Broker und Object Caches beinhaltet.

- *Gatherer*: sammelt Index-Informationen wie Schlüsselwörter, Autorennamen und Titel aus den Quellen, die von Providern zur Verfügung gestellt werden (FTP-, HTTP-Server).
- *Broker*: holt sich Index-Informationen von einem oder mehreren Gatherern, eliminiert mehrfache Informationen, indiziert die gesammelten Informationen und unterstützt eine WWW-Schnittstelle zu den Informationen.
- *Object Cache*: Hält bereits indexierte Informationen für den Nutzer bereit.
- Gatherer und Broker arbeiten unabhängig voneinander, lediglich das Kommunikationsformat ist festgelegt.

Gatherer und Broker können leicht geändert oder ausgetauscht werden.

Architektur von Harvest [BDH+95, BDH+94]



Merkmale von Harvest

- Heterogenität: Auf Brokerseite können unterschiedlichste Textretrievalsysteme eingesetzt werden, Voraussetzung ist, daß es boolesche Kombinationen von attributbasierten Anfragen zuläßt und inkrementelle Änderungen unterstützt. Möglich sind etwa: Ingres, Wais, Glimpse, Nebula, etc.
- Verteiltheit: Die einzelnen Komponenten wie Gatherer, Broker oder Object Cache sind auf verschiedenen Servern installiert. Keine Komponente verfügt über die gesamte Information des Systems. Zentral organisiert ist nur die Harvest Server Registry (HSR), sozusagen ein Broker-Broker.
- Replikation: Replizierung von Brokern ist Bestandteil des Systems; schwache Konsistenz ist durch Benutzen eines spez. Werkzeuges gesichert.
- Verteiltes Retrieval: Über den HSR läßt sich einer der thematisch spezialisierten Broker für eine spezielle Anfrage auswählen, die nur an einen gestellt werden kann; bei Bedarf muß sie an weitere Broker wiederholt gestellt werden.

- Metadatenhaltung: Es stehen nur Metadaten zu Dokumenten zur Verfügung, die anhand einer URL erreichbar sind.

  Eigenes Format für Metadaten: SOIF (Summary Object Interchange Format), bestehend aus einer Liste von Attribut-/Wert-Paaren; die Attribute können vom Gatherer-Betreiber auch neu hinzugefügt werden.
- Automatische Generierung und Aktualisierung durch den Gatherer; der Betreiber des Brokers legt die Wartungsintervalle fest.
- Manuelle Nachbearbeitung der automatisch erstellten SOIF-Dokumente ist möglich.
- Datenbasis-Auswahl: Es wird nur in der Datenbasis des angesprochenen Brokers gesucht. Informationsquellen werden vom Betreiber manuell festgelegt. Dies können Gatherer oder andere Broker sein.
- Dokumentformate: prinzipiell werden alle Formate akzeptiert; der Grad der Verarbeitung hängt stark vom Format ab.
- Logische Dokumentstrukturen: Ein Dokument ist eine Menge von Feldern, hierarchische Strukturierung ist also nur im Rahmen des Dokumentenextrahierers möglich.

- Bedienbarkeit: Sehr unkomfortabel; Der HSR liefert eine Liste von Brokern mit geeigneten Informationen. Bei jedem dieser Broker muß die Anfrage neu gestellt werden.
- Updates: Automatisch, nach Konfiguration.
- Relevance Feedback, Browsing, Profildienste, versch. Datentypen, Multimedialität werden noch nicht unterstützt, sind aber geplant.

11.2 Benutzungsschnittstellen

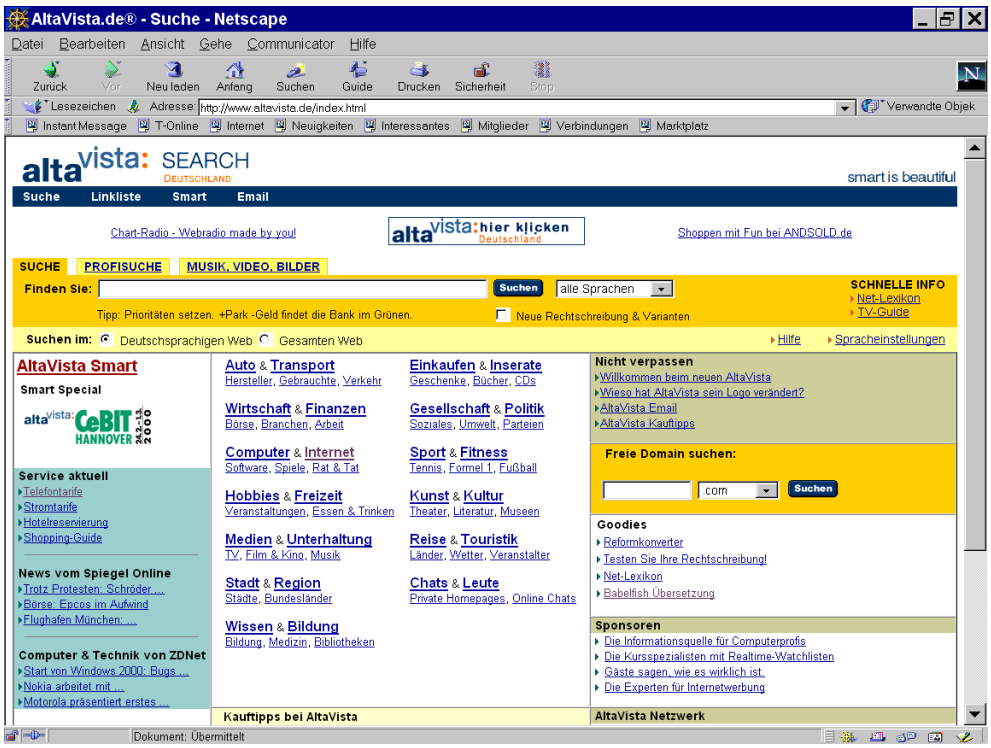
Die Benutzungsschnittstelle muss zwei Aspekte abdecken:

- Die Eingabe der Anfrage
- Die Ausgabe des Suchergebnisses

Im Hinblick auf die Anfrage werden dabei häufig mehrere Möglichkeiten angeboten:

- Verwendung einer einfachen Anfrageformulierung
- Angabe einer stark ausdifferenzierten Anfrage
- Verwendung eines Katalogs (der letztlich einem Klassifikationschema entspricht)

Anfrageschnittstelle von Altavista (hier: einfache Anfrage und Katalog):



Mögliche Anfragekriterien am Beispiel Hotbot (nutzt INKTOMI Search-Engine):

- *Look For*: Search for pages containing the search term(s)in the form specified.
- *Language*: Limit results to a specific language.
- *Word Filter*: Limit results to pages containing/not containing the words specified.
- *Date*: Limit results to pages published within a specified period of time.
- *Pages Must Include*: Return only pages containing the specified media types or technologies.

  image, audio, MP3, video, Shockwave, Java, JavaScript, ActiveX, VRML, Acrobat, VB-Script, Win-Media, RealAudio/Video oder eine vorgegebene extension
- *Location/Domain*: Return only pages in specific domains (wired.com, doj.gov), top-level domains (.edu, .com), and/or specific continents or countries.
- *Page Depth*: Control what types of pages are searched within each Web site.
- *Word Stemming*: Search for grammatical variations of your search term.

  Ex: Searches for „thought“ will also find „think“ and „thinking.“

Search Engines	Boolean	Proximity	Truncation	Case	Fields	Limits	Stop	Sorting
All the Web (Fast)	+, -, **and**	Phrase	No	No	title, URL, link, more	Language, domains	No	Relevance
Lycos Pro (Fast)	+, -, **and**	Phrase	No	No	Title, URL, link, more	Language, domain	No	Relevance
Northern Light	**and**, or, not, (), +, -	Phrase	Yes * %, auto plurals	No	Title,URL, more	Doc type, date, more	No	Custom Folders, date
AltaVista Simple	+, -	**Phrase**	Yes *	Yes	Title, URL, link, more	Language	Yes	AskJeeves, Real-Names, Relevance
AltaVista Adv.	and, **or**, and not, ()	**Phrase**, Near	Yes *	Yes	Title, URL, link, more	Language, date	No	Relevance, if used
Excite	AND, **OR**, NOT, (), +, -	Phrase	No	No	No	Language, domain	Yes	Relevance, site
Google	**and**, -, +	Phrase	No	No	link, related	No	Yes, + searches	Relevance, on citation
Infoseek	+, -	Phrase	No, auto plurals	Yes	Title, URL, link, site, alt	No	No	Relevance, site, date
HotBot	**and**, or, not, (), +, -	Phrase	Yes *	Yes	Title, more	Language, date, more	Yes	Relevance, site
WebCrawler	and, **or**, not, (), +, -	Phrase, near, adj	No	No	No	No	Yes	Relevance

Darstellung der Antwort

Die Antwort zu einer Anfrage besteht meist aus einer geordneten Liste der gefundenen Web-Seiten.

In dieser Liste werden im Allgemeinen zu jedem Dokument einige Informationen angeführt, die vorzugsweise den META-Tags entnommen werden.

Sind keine META-Tags angegeben, werden die ersten Zeilen des Dokumentes verwendet.

Häufig werden auch weitere Informationen wie das letzte Änderungsdatum und die Größe der Datei angezeigt.

11.3 Ranking und Zugriffsstrukturen

Die Suchmaschinen arbeiten im Allgemeinen mit Varianten des Booleschen Retrieval oder des Vektorraummodells.

Die spezifischen Feinheiten sind aber mehr oder weniger Geheim.

In der Regel gehen folgende Informationen in die Gewichtung ein:

- Die Übereinstimmung zwischen Anfrage und Dokument
- Die Wichtigkeit des Dokumentes (z.B. in Anzahl der Zugriffe oder in Anzahl der Links auf die Seite)
- Die Zahlungswilligkeit des Seitenbetreibers

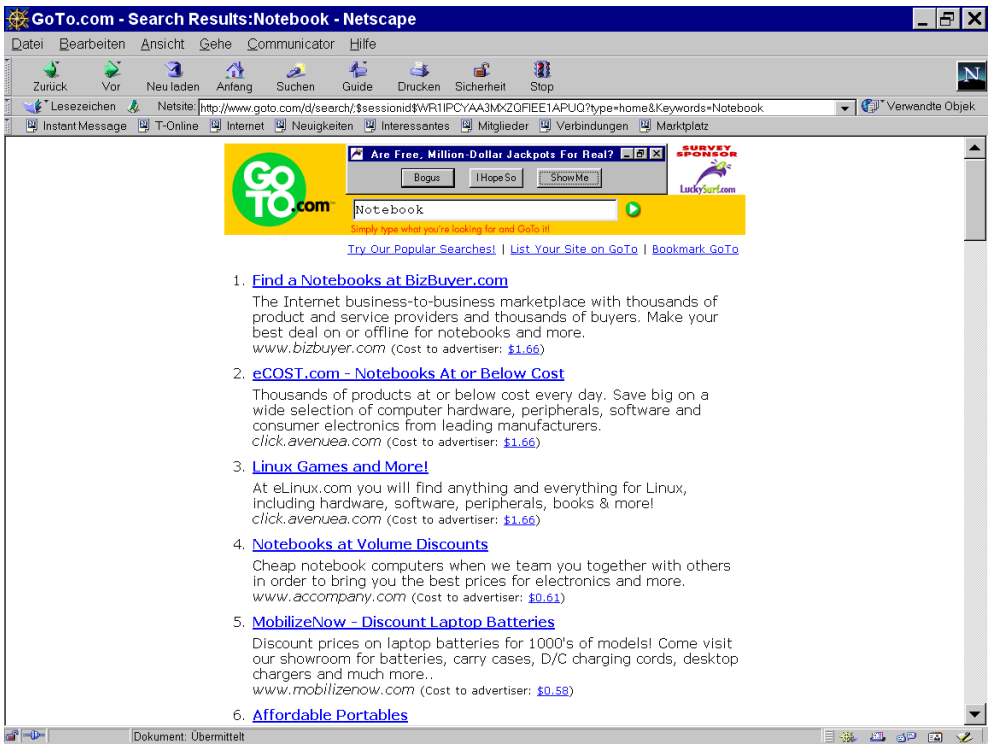
Die Übereinstimmung zwischen Anfrage und Dokument

- Anzahl der zwischen Anfrage und Dokument übereinstimmenden Wörter.
- Häufigkeit des Vorkommens von Suchbegriffen im Dokument.
- Position des Vorkommens.

  Vorkommensorte in der Reihenfolge fallender Gewichtung:
  - Domain und URL
  - Titel
  - Überschrift
  - Meta-Tag (Content oder Keywords)

    Problem: Praxis des Spamming von Meta-Tags → Dokumente, die ein Wort zu häufig im Meta-Tag gelistet haben, werden „bestraft“
  - Dokumentenanfang

Ranking nach dem, was der Seitenbetreiber pro Click zahlt – Beispiel: www.goto.com



Die Maschinen listen aber nicht nur bezahlte Listenplätze, sondern verfügen auch über eine „normale“ Datenbank.

Nur die ersten Treffer sind bezahlte Ränge.

Einer der großen Kritikpunkte dieses Verfahrens: Nur wer ordentlich investiert, kommt nach oben.

Qualität und Inhalt spielen nur eine sehr untergeordnete Rolle.

Weitere Variante: *RealNames*

- Sieht aus wie eine Rankingmethode, ist aber eigentlich keine.
- Realnames ist eine Adressdatenbank, in die sich Unternehmen und Seitenbetreiber kostenpflichtig eintragen können.
- Die Adresse kann mit Stichworten verknüpft werden; sie wird dann bei entsprechenden Suchanfragen zusätzlich zu den Suchtreffern gelistet.

Eine weitere Variante ist besonders oft besuchte Seiten hoch zu ranken nach dem Motto: „Millionen Fliegen können nicht irren“

Beispiel: `http://www.directhit.com`

Ranking nach Verlinkungshäufigkeit

- Ein neuer Weg der Sortierung von Suchergebnissen ist, die Einordnung einer Seite davon abhängig zu machen, wie oft diese von anderer Stelle verlinkt wurde.
- Je mehr Links auf eine Seite verweisen, desto höher der Platz in der Ergebnisliste.
- Die Suchmaschine Google (`www.google.com`) arbeitet nach diesem Prinzip.
- Die Realisierung gestaltet sich aufwendig.
- Ein komplexes Muster von Wertigkeiten muss erfaßt werden, denn neben der Zahl der Links wird auch festgestellt, wie oft die Seite, von der der Verweis stammt, ihrerseits verlinkt ist.
- Ein Link von einer Seite, auf die häufig verwiesen wird, wiegt „schwerer“ als der von einer weniger oft verlinkten Seite.
- Zusätzliche Rankingpunkte gibt es, wenn der Suchbegriff als Linktext gefunden wird.
- Die Erfahrung der Google-Macher zeigte, dass der Text, der mit dem Verweis hinterlegt ist, oft in höherem Maße den Inhalt einer Seite beschreibt als die dort gesammelten Informationen im Titel oder den Meta-Tags.

Die Implementierung der Suche erfolgt dabei in der Regel mit invertierten Listen.

Dabei wird eine umfangreiche Hardware genutzt.

Für Altavista wird z.B. in [Kar99] folgende Konfiguration skizziert:

- Die Alta Vista Indizierungssoftware bewältigt pro Stunde ein Gigabyte Text.
- Der Gesamtindex beträgt 40 Gigabyte.
- 3 Millionen Seiten werden täglich gescannt.
- Die 16 Maschinen, auf denen die Indizierung und Suchabfrage stattfindet, arbeiten mit je 8 Gigabyte Arbeitsspeicher, 10 parallelen Alpha-Prozessoren, und 260 GB Festplattenkapazität.
- Die Vorkommunikation ins Netz erledigen ein paar andere Rechner: Mehrere Alpha-Rechner mit 256 MB Arbeitsspeicher und 4 Gigabyte Festplatten bedienen die Anfragen.

11.4 Kataloge

Neben der Suche werden auch Kataloge zum Web angeboten.

Diese sind im Prinzip mit Klassifikationen zu vergleichen.

11.5 Meta-Suchmaschinen

Eine andere Entwicklung stellen Meta-Suchmaschinen dar, die selbst keinen Index verwalten, sondern auf die Ergebnisse anderer Suchmaschinen zurückgreifen.

Die Ergebnisse der anderen Suchmaschinen können dabei ggf. auch im Sinne von Inferenznetzwerken verbunden werden.

Quellen zu Suchmaschinen im Internet:

- Home of Search Engine Showdown (`http://www.notess.com`)
- Search Engine Watch (`http://www.searchenginewatch.com`)
- Die Homepage zu [Kar99]: `http://suchfibel.de`
- `http://www.kcpl.lib.mo.us/search/srchengines.htm`

Literaturverzeichnis

[BC94] J. Broglio and W. B. Croft. Query processing for retrieval from large text bases. In *Proc. ARPA Human Language Technology Workshop '93*, pages 353–357, Princeton, NJ, March 1994. distributed as *Human Language Technology* by San Mateo, CA: Morgan Kaufmann Publishers.

[BDH+94] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, Michael F. Schwartz, and Duane P. Wessels. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, 1994.

[BDH+95] C. M. Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28(1-2):119–125, December 1995.

[Bec97] Peter Becker. Information Retrieval – Datenstrukturen und algorithmische Grundlagen. Vorlesungsskript, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Tübingen, 1997.

[BL85] C. Buckley and A. Lewit. Optimization of inverted vector searches. In *Proceedings of the 8th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 97–105, New York, 1985.

[BSM97] Ilja N. Bronstein, Konstantin A. Semendjajew, and Gerhard Musiol. *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt/Main, 1997.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley-Longman, May 1999.

[Cho99] G.G. Chowdhury. *Introduction to Modern Information Retrieval*. Library Association Publishing, London, 1999.

[CMK66] Cyril Cleverdon, Jack Mills, and Michael Keen. *Factors Determining the Performance of Indexing Systems: ASLIB Cranfield Research Project. Volume 1: Design.* ASLIB Cranfield Research Project, Cranfield, 1966.

[Cro90] J.C. Crouch. An approach to the automatic construction of global thesauri. *Information Processing and Management*, 26(5):629–640, 1990.

[Cro98] W. B. Croft, editor. *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Austrailien, August 1998. ACM.

[Daw74] J.L. Dawson. Suffix Removal and Word Conflation. *ALLC Bulletin*, pages 33–46, 1974.

[DDF+90] Scott Deerwester, Susan Dumais, Goerge Furnas, Thomas Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[FB91] Norbert Fuhr and Chris Buckley. A probabilistic learning approach for document indexing. *ACM Transactions on Information Systems*, 9(3):223–248, July 1991. Special Issue on Research and Development in Information Retrieval.

[FBY92] William B. Frakes and Ricardo Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1992.

[Fer99] Reginald Ferber. Data Mining und Information Retrieval. Vorlesungsskript (`http://www-cui.darmstadt.gmd.de/~ferber/dm-ir`), Technische Hochschule Darmstadt, Darmstadt, Dezember 1999.

[FMS91] H. P. Frei, S. Meienberg, and P. Schäuble. The perils of interpreting recall and precision values. In Norbert Fuhr, editor, *Information Retrieval: GI/GMD-Workshop Darmstadt*, pages 1–10. Springer Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1991.

[FR97] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1):32–66, January 1997.

[Fra92] William B. Frakes. *Stemming algorithms*, pages 131–160. In Frakes and Baeza-Yates [FBY92], 1992.

[Fuh90] Norbert Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *16th International Conference on Very Large Data Bases*, pages 696–707, Brisbane, Queensland, Australia, 13–16 August 1990. Morgan Kaufmann.

[Fuh96] Norbert Fuhr. Skriptum Information Retrieval. Lecture Notes on Information Retrieval - Universtität Dortmund (`http://ls6-www.informatik.uni-dortmund.de/ir/teaching/courses/ir`), 1996.

[GG98] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[Gul88] J. Gulbins. *UNIX: Eine Einführung in Begriffe und Kommandos*. Springer Compass. Springer-Verlag, 3 edition, 1988.

[Hea99] Marti Hearst, editor. *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Berkeley, CA, USA, August 1999. ACM.

[Hen94] A. Henrich. A distance-scan algorithm for spatial access structures. In N. Pissinou and K. Makki, editors, *Proceedings of the 2nd ACM Workshop on Advances in Geographic Information Systems (GIS '94)*, pages 136–143, Gaithersburg, Md., USA, December 1994.

[Hen96] A. Henrich. Document retrieval facilities for repository-based system development environments. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 101–109, Zürich, 1996.

[HM95] A. Henrich and J. Möller. Extending a spatial access structure to support additional standard attributes. In M.J. Egenhofer and J.R. Herring, editors, *Proceedings of the 4th International Symposium on Advances in Spatial Databases (SSD '95)*, volume 951 of *Lecture Notes in Computer Science*, pages 132–151, Portland, ME, USA, August 1995. Springer.

[HS95] G.R. Hjaltason and H. Samet. Ranking in spatial databases. In M.J. Egenhofer and J.R. Herring, editors, *Proceedings of the 4th International Symposium on Advances in Spatial Databases (SSD '95)*, volume 951 of *Lecture Notes in Computer Science*, pages 83–95, Portland, ME, USA, August 1995. Springer.

[HSW89] A. Henrich, H.-W. Six, and P. Widmayer. The LSD tree: Spatial access to multidimensional point and nonpoint objects. In P.M.G. Apers and G. Wiederhold, editors, *Proceedings of the Fifteenth International Conference on Very Large Data Bases*, pages 45–53, Amsterdam, August 1989. Morgan Kaufmann.

[Kar99] Stefan Karzauninkat. *Die Suchfibel – Wie findet man Informationen im Internet?* Ernst Klett Verlag, Leipzig, zweite überarbeitete und ergänzte auflage edition, 1999. (weitere Informationen unter `www.suchfibel.de`).

[Knu73] D. E. Knuth. *The Art of Computer Programming III: Sorting and Searching.* Addison–Wesley, Reading, Massachusetts, 1973.

[Kow97] Gerald Kowalski. *Information Retrieval Systems: Theory and Implentation.* Kluver Academinc Publishers, Boston / Dordrecht / London, 1997.

[Kuh77] Rainer Kuhlen. *Experimentelle Morphologie in der Informationswissenschaft.* Verlag Dokumentation, München, 1977.

[Kuh90] Rainer Kuhlen. Zum Stand pragmatischer Forschung in der Informationswissenschaft. In J. Herget and R. Kuhlen, editors, *Pragmatische Aspekte beim Entwurf und Betrieb von Informationssystemen. Proceedings des 1. Internationalen Symposiums für Informationswissenschaft*, pages 13–18, Konstanz, 1990. Universitätsverlag Konstanz.

[Lan87] Langenscheidts Großes Schulwörterbuch Englisch-Deutsch (17. Auflage). Langenscheid KG, Berlin und München, 1987.

[Lov68] Julie Beth Lovins. Developing of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1), March 1968.

[Mea91] Charles T. Meadow. *Text Information Retrieval Systems*. Academic Press Inc., Harcourt Brace Jovanovich, Publishers, Sab Diego, 1991.

[Mei97] Brigitte Meiss. *Information Retrieval und Dokumentenmanagement im Multimedia-Zeitalter*. Reihe Informationswissenschaft der DGD. Deutsche Gesellschaft für Dokumentation, Frankfurt am Main, 1997.

[Mit95] B. Mitschang. *Anfrageverarbeitung in Datenbanksystemen: Entwurfs- und Implementierungskonzepte*. Datenbanksysteme. Vieweg, Braunschweig, Wiesbaden, 1995.

[MM97] James Mayfield and Paul McNamee. N-Grams vs. Words as Indexing Terms. In *TREC-6 Conference Notebook Papers*, Gaithersburg, Maryland, USA, 1997. `http://www.cs.umbc.edu/~mayfield/pubs/APL-TREC97prelim.ps`.

[Noh99] Holger Nohr. Verbale Inhaltserschließung Indexierung. Vorlesungsskript (`http://www.hbi-stuttgart.de/nohr/thesaurus/index.htm`), Hochschule für Bibliotheks- und Informationswesen, Stuttgart, Januar 1999.

[OMK91] Y. Ogawa, T. Morita, and K. Kobayashi. A fuzzy document retrieval system using the keyword connection matrix and a learning method. *Fuzzy Sets and Systems*, 39:163–179, 1991.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.

[Por80] M. F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.

[PP97] Ulrich Pfeifer and Stefan Pennekamp. Incremental Processing of Vague Queries in Interactive Retrieval Systems. In Norbert Fuhr, Gisbert Dittrich, and Klaus Tochtermann, editors, *Hypertext - Information Retrieval - Multimedia '97: Theorien, Modelle und Implementierungen integrierter elektronischer Informationssysteme*, pages 223–235, Dortmund, Oktober 1997. Universitätsverlag Konstanz.

[Rij79] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 2 edition, 1979.

[RJ76] S. E. Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Sciences*, 27(3):129–146, May-June 1976.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In M.J. Carey and D.A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 71–79, San Jose, Cal., USA, May 1995.

[SB88] G. Salton and C. Buckley. Term-weighted approaches to automatic text retrieval. *Inf. Proc. & Mngmt.*, 24(5):513–523, ? 1988.

[SB90] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *J. Amer. Soc. for Information Sci.*, 41(4):288–297, 1990.

[SBM96] Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Experimental Studies, pages 21–29, 1996.

[SM83] G. Salton and M.J. McGill. *Information Retrieval-Grundlegendes für Informationswissenschaftler*. McGraw-Hill Book Company GmbH, 1983.

[SP93] Michael Schmidt and Ulrich Pfeifer. Eignung von Signaturbäumen für Best-Match-Anfragen. In Gerhard Knorz, Jürgen Krause, and Christa Womser-Hacker, editors, *Information Retrieval '93: Von der Modellierung zur Anwendung, Proceedings der 1. Tagung „Information Retrieval"*, pages 125–138, Regensburg, September 1993. Schriften zur Informationswissenschaft 12, Universitätsverlag Konstanz, 1993, ISBN 3-87940-473-9.

[Str94] Tomek Strzalkowski. Robust text processing in automated information retrieval. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing (13–15 October 1994, Stuttgart)*. Association for Computational Linguistics, October 1994.

[TC90] Howard Turtle and W. Bruce Croft. Inference networks for document retrieval. In *Proceedings of the Thirteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Information Retrieval Models (1), pages 1–24, 1990.

[TC91] Howard Turtle and W. Bruce Croft. Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222, 1991.

[VH98] Ellen M. Voorhees and Donna Harman. Overview of the Seventh Text REtrieval Conference (TREC-7). In *Proceedings of the Seventh Text REtrieval Conference (TREC-7)*, pages 1–24, Gaithersburg, Maryland, USA, 1998. NIST Special Publication 500-242. `http://trec.nist.gov/pubs/trec7/t7_proceedings.html`.

[Vic70] B.C. Vickery. *Techniques of information retrieval*. Butterworth, London, 1970.

[WZRW89] S. K. M. Wong, Wojciech Ziarko, Vijay V. Raghavan, and P. C. N. Wong. Extended boolean query processing in the generalized vector space model. *Information Systems*, 14(1):47–63, 1989.

[WZW85] S. K. M. Wong, W. Ziarko, and P. C. N. Wong. Generalized vector space model in information retrieval. In *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 18–25, New York, USA, 1985.