Information Retrieval
Skriptum zur Vorlesung

Norbert Fuhr

28. April 1998

Inhaltsverzeichnis

1 Einführung **7**

1.1 Textretrieval am Beispiel von Literaturdatenbanken 8

1.1.1 Herkömmliche IR-Systeme 8

1.1.2 Boolesches Retrieval vs. Ranking 10

1.2 IR in anderen Datenbanken 11

1.2.1 Vage Anfragen in Faktendatenbanken 11

1.3 IR in komplexen technischen Anwendungsgebieten 12

2 IR-Konzepte **14**

2.1 Definition von Information Retrieval 14

2.2 Daten — Wissen — Information 15

2.3 Grundmodell des Information Retrieval 17

2.4 Konzeptionelles Modell für IR-Systeme 17

3 Evaluierung **23**

3.1 Effizienz und Effektivität 23

3.2 Relevanz 24

3.3 Distributionen 24

3.4 Standpunkte und Bewertungsmaße 25

3.4.1 Benutzerstandpunkte 25

3.4.2 Benutzer- vs. Systemstandpunkte 26

3.5 Maße für boolesches Retrieval 26

3.5.1 Mittelwertbildung 28

3.6 Rangordnungen 29

3.6.1 Lineare Ordnung 30

3.6.2 Schwache Ordnung 31

3.7 Interpretation von Recall-Precision-Graphen 35

3.7.1 Abbruchkriterium: Anzahl der relevanten Dokumente (NR) . . 35

3.7.2 Abbruchkriterium: Anzahl der Dokumente 38

3.8 Mittelwertbildung und Signifikanztests bei Rangordnungen 38

3.9 Nützlichkeitsmaß 44

4 Wissensrepräsentation für Texte **49**
4.1 Problemstellung 49
4.2 Freitextsuche 50
4.2.1 Informatischer Ansatz 51
4.2.2 Computerlinguistischer Ansatz 52
4.3 Dokumentationssprachen 59
4.3.1 Allgemeine Eigenschaften 59
4.3.2 Klassifikationen 59
4.3.3 Thesauri 66
4.3.4 Semantische Netze am Beispiel KL-ONE 72
4.3.5 Dokumentationssprachen vs. Freitext 78
4.4 Beurteilung der Verfahren zur Repräsentation von Textinhalten . . . 78
4.5 Zusammenhang zwischen Modellen und Repräsentationen 79
4.5.1 Textrepräsentation für IR-Modelle 79
4.5.2 Einfache statistische Modelle 79
4.5.3 Vorkommensformen 80

5 Nicht-probabilistische IR-Modelle **83**
5.1 Notationen 83
5.2 Überblick über die Modelle 84
5.3 Boolesches Retrieval 85
5.3.1 Mächtigkeit der booleschen Anfragesprache 86
5.3.2 Nachteile des booleschen Retrieval 86
5.4 Fuzzy-Retrieval 87
5.4.1 Beurteilung des Fuzzy-Retrieval 88
5.5 Das Vektorraummodell 89
5.5.1 Relevance Feedback 90
5.5.2 Indexierung in SMART 92
5.5.3 Beurteilung des VRM 94
5.6 Dokumenten-Clustering 94
5.6.1 Cluster-Retrieval 95
5.6.2 Ähnlichkeitssuche von Dokumenten 96
5.6.3 Cluster-Browsing 96
5.6.4 Scatter/Gather-Browsing 97

6 Probabilistic Models in Information Retrieval **99**
6.1 Introduction 99
6.2 Basic concepts of relevance models 100
6.2.1 The binary independence retrieval model 100
6.2.2 A conceptual model for IR 103
6.2.3 Parameter learning in IR 104

6.2.4 Event space . . . 106
6.2.5 The Probability Ranking Principle . . . 107
6.3 Some relevance models . . . 108
6.3.1 A description-oriented approach for retrieval functions . . . 108
6.3.2 The binary independence indexing model . . . 115
6.3.3 A description-oriented indexing approach . . . 116
6.3.4 The 2-Poisson model . . . 119
6.3.5 Retrieval with probabilistic indexing . . . 120
6.4 IR as uncertain inference . . . 123
6.4.1 Imaging . . . 125
6.4.2 Inference networks . . . 125
6.5 General problems . . . 127
6.5.1 Parameter estimation . . . 127
6.5.2 Query expansion . . . 135
6.5.3 Representation of documents and queries . . . 135
6.6 Conclusion and outlook . . . 137

7 Models based on predicate logic 138
7.1 Introduction . . . 138
7.2 Terminological logic . . . 139
7.2.1 Thesauri . . . 139
7.2.2 Elements of terminological logic . . . 140
7.2.3 Semantics of MIRTL . . . 142
7.2.4 Retrieval with terminological logic . . . 143
7.3 Datalog . . . 146
7.3.1 Introduction . . . 146
7.3.2 Hypertext structure . . . 147
7.3.3 Aggregation . . . 148
7.3.4 Object hierarchy . . . 149
7.3.5 Retrieval with terminological knowledge . . . 150
7.4 Probabilistic Datalog . . . 152
7.4.1 Introduction . . . 152
7.4.2 Informal description of Datalog$_P$ . . . 153
7.4.3 Syntax . . . 155
7.4.4 Semantics of Datalog$_P$ . . . 156
7.4.5 Evaluation of Datalog$_P$ programs . . . 159
7.4.6 Datalog$_P$ with independence assumptions . . . 161
7.4.7 Further application examples . . . 162
7.4.8 Probabilistic rules . . . 164

8 Implementierung von IR-Systemen 166
8.1 Hardware-Aspekte . . . 166
8.1.1 Speichermedien . . . 166
8.1.2 Ein-/Ausgabegeräte . . . 168
8.1.3 Kommunikationsnetzwerke . . . 168
8.2 Aufbau von IRS . . . 169
8.2.1 Funktionale Sicht . . . 169
8.2.2 Dateistruktur . . . 169
8.2.3 Dialogfunktionen herkömmlicher IRS . . . 170
8.3 Dokumentarchitekturen . . . 171
8.3.1 ODA . . . 171
8.4 Zugriffspfade . . . 181
8.4.1 Scanning . . . 181
8.4.2 Ähnlichkeit von Zeichenketten . . . 194
8.4.3 Invertierte Listen . . . 195
8.4.4 Signaturen . . . 201
8.4.5 PAT-Bäume . . . 230

9 IR and Databases 240
9.1 Introduction . . . 240
9.2 Basic DBMS features . . . 242
9.2.1 Data model . . . 242
9.2.2 Integrity . . . 242
9.2.3 Recovery . . . 244
9.2.4 Concurrency . . . 244
9.2.5 Security . . . 245
9.3 Modeling structure . . . 246
9.3.1 Relational model . . . 246
9.3.2 NF^2 model . . . 249
9.4 Object-oriented databases . . . 252
9.4.1 Modeling structure . . . 252
9.4.2 Modeling behaviour . . . 257

10 Fact Retrieval 259
10.1 A Probabilistic Approach to Fact Retrieval . . . 259
10.1.1 Introduction . . . 259
10.1.2 Foundations of the probabilistic model . . . 262
10.1.3 Indexing for missing or imprecise data . . . 266
10.1.4 Retrieval functions . . . 270
10.2 Integration of Text and Fact Retrieval . . . 271
10.2.1 Introduction . . . 271

10.2.2 Extending text retrieval methods for coping with facts 271
10.2.3 An application example 273

11 Interaktive Informationssysteme **277**
11.1 Funktionen für interaktive Informationssysteme 277
11.2 Zusammenspiel zwischen System und Benutzer 279
11.2.1 Stufen der Systembeteiligung 280
11.2.2 Arten von Suchaktivitäten 281
11.2.3 Kombination von Systembeteiligung und Suchaktivitäten . . . 281

Vorwort

Das vorliegende Skript dient als Begleitlektüre zur vierstündigen Spezialvorlesung „Information Retrieval“. Da es leider zur Zeit kein wirklich empfehlenswertes Lehrbuch zu diesem Thema gibt, wird hier der Versuch einer Zusammenstellung relevanter Themen aus diesem Gebiet gemacht.
An dieser Stelle möchte ich all jenen danken, die am Zustandekommen dieses Skriptums mitgewirkt haben. Das Kapitel über Evaluierung wurde in wesentlichem Maße von Ulrich Pfeifer gestaltet. Ein Teil des Abschnitts über Signaturen basiert auf einer Vorlage von Renato Vinga. Tung Huynh und Thorsten ernst haben zahlreiche Abbildungen erstellt und als TeXperten die Erstellung der endgültigen schriftlichen Fassung (wie auch schon der Folien zur Vorlesung) übernommen; hierfür möchte ich ihnen sehr herzlich danken. Frank Deufel, Kostadinos Tzeras, Oliver Stach, Jörg Schneider und Jürgen Kalinski gebührt Dank für ihre sorgfältige Lektüre vorheriger Fassungen dieses Skriptums, die zur Korrektur zahlreicher Fehler führten.

Literatur

Wie eingangs erwähnt, gibt es kaum geeignete Lehrbücher zum Gebiet des Information Retrieval. Insbesondere sind alle in Deutsch erschienenen Bücher zu diesem Thema völlig unbrauchbar. Bedingt empfehlenswert sind die Bücher von Robert Korfhage [Korfhage 97] und Frakes/Baeza-Yates [Frakes & Baeza-Yates 92], wobei letzteres sich nur auf die implementierungstechnischen Aspekte von Textretrievalsystemen beschränkt (siehe hierzu auch [Witten et al. 94]). Als vertiefende Literatur sind die von Sparck Jones und Willet zusammengestellten „Readings in IR“ [Sparck Jones & Willet 97] sehr zu empfehlen. Immer noch lesenswert ist auch das Buch von Rijsbergen [Rijsbergen 79a], das in elektronischer Form unter `http://www.dcs.glasgow.ac.uk/Keith/Preface.html` verfügbar ist.
Literaturhinweise zu den einzelnen behandelten Themen finden sich im Text. Wo sich (wie angegeben) längere Abschnitte auf einzelne Publikationen stützen, ist natürlich die Lektüre der Originalarbeit zu empfehlen. Als Quelle für Beiträge aus der aktuellen IR-Forschung sind zuallererst die Tagungsbände der jährlich stattfindenden *ACM-SIGIR Conference on Research and Development in Information Retrieval* zu nennen. Auch die alle drei Jahre stattfindende RIAO-Tagung (Recherche d'informations assisté par Ordinateur) befaßt sich schwerpunktmäßig mit IR und verwandten Gebieten. Ganz oder in wesentlichen Teilen widmen sich dem Gebiet des IR die Zeitschriften *ACM Transactions on Information Systems*, *Information Processing and Management*, *Journal of the American Society for Information Science*, *Journal of Documentation* und *Nachrichten für Dokumentation*. Die drei letztgenannten enthalten auch zahlreiche Berichte aus der Praxis, ebenso die Zeitschriften *cogito* und *INFOdoc*.

Kapitel 1

Einführung

Will man den Gegenstand des Information Retrieval mit wenigen Worten beschreiben, so ist die Formulierung „inhaltliche Suche in Texten“ wohl am treffendsten. Tatsächlich wird damit aber nur ein wesentlicher — wenn auch der wichtigste — Bereich des Information Retrieval umschrieben, den man auch häufig als Textretrieval oder Dokumentenretrieval bezeichnet. In 2.1 werden wir eine wesentlich weitere Definition des Gegenstandsbereiches von IR kennenlernen, doch wollen wir uns hier zunächst auf das Textretrieval beschränken.

Das klassische Anwendungsgebiet des Textretrieval sind Literaturdatenbanken, in denen Kurzfassungen von Veröffentlichungen gespeichert werden, und die den Anwendern die Suche nach für sie relevanter Literatur in einem bestimmten Fachgebiet ermöglichen sollen (siehe nächster Abschnitt). Durch den technologischen Fortschritt im Bereich der Rechnerhardware, insbesondere bei den Speichermedien, sind aber neue Anwendungen hinzugekommen, die werden – gemessen an ihrer Verbreitung – wenn nicht schon heute, so doch kurz- bis mittelfristig wesentlich wichtiger für das Gebiet des IR. Als zusätzlicher Faktor ist hier natürlich auch die heute fast ausschließlich rechnerbasierte Erstellung von Dokumenten zu nennen, die es natürlich nahelegt, die einmal erstellten Dokumente dann auch auf dem Rechner zu speichern, zu verbreiten (z.B. als “electronic mail”) und bei Bedarf abzurufen, statt die Dokumente auszudrucken und dann mit konventionellen Mitteln weiterzuarbeiten. Bedeutende Anwendungsbereiche für derart erstellte Dokumente sind zum einen die technische Dokumentation, zum anderen Büroinformationssysteme. Im ersten Fall gehen immer mehr Hersteller von komplexen technischen Systemen (z.B. bei Rechnern, etwa das UNIX-Manual) dazu über, die Dokumentation in maschinenlesbarer Form auszuliefern (z.B. auf CD-ROM), weil dies zum einen Kosten sparen kann, zum anderen auch in der Handhabung beim Anwender wesentliche Vorteile mit sich bringt (das aufwendige Austauschen von Handbuchseiten bei neuen Versionen entfällt, mehrere Benutzer haben gleichzeitigen Zugriff auf das Handbuch, der Zugriff ist schneller). Im Bereich der Büroinformationssysteme möchte man vor allem Kosten für die Ablage sparen und

die Kommunikation beschleunigen. Bei beiden Anwendungsbereichen stellt sich aber das Problem, wie der Benutzer bei der Informationssuche effektiv unterstützt werden kann. Es ist offensichtlich, daß in den meisten Fällen die Suche an inhaltlichen Kriterien ausgerichtet ist. Herkömmliche Datenbanksysteme sind für diese Aufgabenstellung denkbar ungeeignet, wie wir im folgenden noch sehen werden.

1.1 Textretrieval am Beispiel von Literaturdatenbanken

1.1.1 Herkömmliche IR-Systeme

```
AN  87(01):411    NTIS Order Number : AD-A172 502/7/XAD
TI  Controlling Inference. (Doctoral thesis)
AU  Smith, David E.
CS  Stanford Univ., CA. Dept. of Computer Science
NC  Contract : N00014-81-K-0004
NR  AD-A172 502/7/XAD; STAN-CS-86-1107
    197 p.    NTIS Prices : MF A01
    Availability : Microfiche copies only.
PD  860400
LA  English    CY  United States
OS  GRA&I8701
AB  Effective control of inference is a fundamental problem in Artificial
    Intelligence. Unguided inference leads to a combinatoral explosion of
    facts or subgoals for even simple domains. To overcome this problem,
    expert systems have used powerful domain-dependent control...
CC   95F  Bionics and artificial intelligence
CT  *Artificial intelligence; Data bases; Decision making; Global;
    Information exchange; Problem solving; Efficiency
    *Inference; Control; Theses; Expert systems
UT  NTISDODXA
```

Abbildung 1.1: Beispiel-Dokument aus der NTIS-Datenbank

Abbildung 1.1 zeigt ein Dokument aus einer Literaturdatenbank, der NTIS-Datenbank, die Daten über Forschungsberichte von staatlich geförderten Projekten in den USA enthält. Das Dokument enthält zum einen bibliographische Angaben, die es ermöglichen, bei Bedarf das Originaldokument eindeutig zu identifizieren und zu beschaffen. Darüber hinaus ist zur Unterstützung der inhaltlichen Suche zum einen ein Abstract vorhanden, zum anderen ist das Dokument klassifiziert (Feld CC) und indexiert (Feld

No	Records	Request
1.	19248	DRUGS
2.	2412	DRUGS in TI
3.	2560	AGING
4.	19119	DRUG not AGING
5.	2349	#2 and #4
6.	9305	MEMORY
7.	6	#5 and (DRUG near4 MEMORY)
8.	22091	COGNITIVE
9.	16	#5 and (DRUG near4 COGNITIVE)
10.	22	#7 or #9
11.	2023	SIDE-EFFECTS-DRUG in DE
12.	0	#11 and #10

Tabelle 1.1: Beispielrecherche zum Thema "The side effects of drugs on memory or cognitive abilities, not related to aging"

CT). Solche Literatur–datenbanken enthalten heute in der Größenordnung 10^5 bis 10^7 Dokumente. Für die Informatik wichtige Datenbanken sind insbesondere die INSPEC-Datenbasis (die zusätzlich das Gebiet der Elektrotechnik abdeckt) und die Datenbasis CompuScience, an deren Aufbau die ACM maßgeblich beteiligt ist.

Wie kann nun eine inhaltliche Suche in einer derartigen Datenbank vonstatten gehen? Aus informatischer Sicht ist es naheliegend, hierzu eine boolesche Anfragesprache zu verwenden, bei der die einzelnen Operanden jeweils Bedingungen an die gesuchten Dokumente spezifizieren. Wie eine derartige Suche abläuft, zeigt das Beispiel in Tabelle 1.1 (der Operator *not* steht hier für 'und nicht', *near4* bedeutet: maximal vier Wörter im Text zwischen den beiden als Operanden angegebenen Wörtern, *in DE* steht für die ausschließliche Suche in der Indexierung der Dokumente). Deutlich sind die Probleme zu erkennen, die Anfrage so zu formulieren, daß man eine vernünftige Anzahl Antworten (etwa zwischen 20 und 100) erhält, von denen man dann die Abstracts liest, um die interessant erscheinenden Dokumente anschließend zu beschaffen. Der Benutzer hat hier z.B. eine gewisse Wahlfreiheit, welche Begriffe er nur mit UND verknüpft oder wo er zusätzliche Abstandsbedingungen formuliert. Enttäuschend ist das Schlußergebnis, daß nämlich kein einziges Dokument existiert, das die logisch korrekte Suchfrage erfüllt (obwohl davon auszugehen ist, daß es wohl mehrere solche Dokumente in der Datenbank gibt). Es stellt sich somit die Frage, ob die Schwierigkeiten in diesem Beispiel (die typisch für Literaturrecherchen sind) immanent für IR-Anwendungen sind, oder ob nicht auch Lösungen möglich sind, die den Anwender leichter zum Ziel gelangen lassen.

1.1.2 Boolesches Retrieval vs. Ranking

Schon in [Verhoeff et al. 61] wird darauf hingewiesen, daß boolesche Anfragesprachen denkbar ungeeignet sind für Textretrieval (siehe auch [Salton et al. 83]). Zum einen ist die fehlende Benutzerfreundlichkeit aufgrund der komplizierten Anfragesprache zu nennen, die die meisten (potentiellen) Nutzer eines solchen Systems überfordert. Weitaus schwerwiegender ist aber die Tatsache, daß bei experimentellen Untersuchungen boolesches Retrieval mit Abstand die schlechteste Retrievalqualität aller bekannten Retrievalverfahren liefert ([Salton et al. 83], [Fox & Koll 88]). Umso erstaunlicher ist daher, daß bis heute fast alle kommerziell angebotenen IR-Systemen ausschließlich boolesches Retrieval verwenden!
Die Ursache für diese schlechte Retrievalqualität des booleschen Retrieval ist wohl darin zu sehen, daß zwei wesentliche Eigenschaften des Textretrieval von dieser Methode ignoriert werden, nämlich

- die Unsicherheit der Repräsentation von Textinhalten: Im Gegensatz zu herkömmlichen Datenbanksystemen mit präzisen Attributwerten lassen sich Textinhalte nie in perfekter Art und Weise auf eindeutige Bezeichner abbilden, unabhängig davon, ob es sich dabei um Wörter aus dem Text, Begriffe aus einem vorgegebenen Vokabular oder um einen anspruchsvollen wissensbasierten Formalismus wie Frames, Logik oder semantische Netzwerke handelt.
- Die Vagheit der Anfragen: Zum einen gilt hier in ähnlicher Weise das oben gesagte bezüglich der Repräsentation von Inhalten, zum anderen sind die im IR betrachteten Informationsprobleme gerade von so komplexer Art, daß sie sich nur schrittweise präzisieren lassen (wie soll man genau wissen, was man nicht weiß?).

Frage	Gewicht	d_1	d_2	d_3	d_4
side effect	1	x	x	x	x
drugs	1	x	x	x	x
memory	1	x		x	
cognitive ability	1		x	x	x
aging	-1		x		
Retrievalgewicht		3	2	4	3

Tabelle 1.2: Beispiel für "coordination level match"

Eine einfache Alternative zum booleschen Retrieval (sowohl für den Anwender als auch für den Systementwickler) stellt der sogenannte "coordination level match" dar. Dabei gibt der Benutzer einfach seine Anfrage in Form einer Menge von Suchwörtern ein. In der einfachsten Form des "coordination level match" wird für jedes Dokument einfach die Anzahl Suchbegriffe aus der Anfrage, die darin vorkommen, gezählt. Dieses Gewicht eines Dokumentes bezüglich einer Anfrage bezeichnet man im Englischen

als "retrieval status value"; wir wollen hierfür die Bezeichnung „Retrievalgewicht" verwenden. Anschließend werden die Dokumente nach der fallenden Anzahl Treffer geordnet ausgegeben (sogenanntes "Ranking"). Tabelle 1.2 zeigt die Vorgehensweise zur Beispielfrage aus dem vorigen Abschnitt. (Als Erweiterung haben wir hier angenommen, daß die Negation durch das Gewicht -1 für das Wort "aging" realisiert wird; das läßt sich auch dadurch plausibel machen, daß es ja Dokumente geben kann, die sowohl das gewünschte Thema als auch den nicht gewünschten Aspekt "aging" beinhalten, und die man deswegen nicht von der Antwort ausschließen sollte). Die Ordnung der Dokumente nach fallendem Retrievalgewicht bietet mehrere Vorteile im Vergleich zum booleschen Retrieval mit seinen ungeordneten Antwortmengen:

- Durch die Rangordnung stehen die am ehesten relevanten Dokumente vermehrt am Anfang der Folge. Wir werden später theoretische Modelle (insbesondere probabilistischer Art) kennenlernen, die diese Behauptung rechtfertigen.
- Im Gegensatz zum booleschen Retrieval gibt es keine Probleme mit der Anzahl der Antworten: Der Benutzer kann die Rangordnung so weit durchsehen, wie er das für sinnvoll hält; er muß nicht umständlich die Anfrage reformulieren, um sich eine bestimmte Anzahl Dokumente anschauen zu können.
- In [Salton et al. 83] beschriebene Experimente zeigen bereits für dieses einfache Verfahren eine bessere Retrievalqualität als beim booleschen Retrieval.

1.2 IR in anderen Datenbanken

Im folgenden wollen wir anhand von zwei Beispielen aufzeigen, daß man IR-Methoden (insbesondere solche, die sich mit Unsicherheit und Vagheit beschäftigen) auch bei anderen Arten von Datenbanken anwenden kann.

1.2.1 Vage Anfragen in Faktendatenbanken

	CPU	MEM	CLOCK_ RATE	DISK_ SIZE	ACC._ TIME	PRICE
A	80386	4	20	40	40	1500
B	80386	4	25	80	28	2000
C	80386	4	25	75	26	2000
D	80386	4	25	80	24	3000
E	80386	4	25	85	28	2500

Tabelle 1.3: Beispiel-Datenbank über PCs

Tabelle 1.3 stellt den Auschnitt einer Datenbank mit Kenndaten von am Markt angebotene PCs dar. Ein Benutzer stellt nun die Anfrage: „Suche kostengünstigen

386er-PC mit mindestens 25 MHZ und einer 80 MB-Festplatte mit maximal 25ms Zugriffszeit". Es ist offensichtlich, daß eine strenge Interpretation der vom Benutzer genannten Kriterien nicht adäquat ist, weil diese der Natur nach keine „harten" Anforderungen, sondern eher vage Zielvorstellungen sind. Wie man anhand des Datenbankinhalts erkennt, kann der Anwender viel Geld sparen, wenn die genannten Kriterien nur geringfügig modifiziert werden. Ein System für eine solche Anwendung sollte dementsprechend die Kriterien als vage interpretieren und zusätzlich zu den korrekten Antworten auch solche ausgeben, die diesen nahekommen. Im anderen Falle wäre ein Benutzer gezwungen, seine Anfrage mehrfach zu modifizieren, um diese zusätzlichen, für ihn interessanten Antworten zu erhalten.

1.3 IR in komplexen technischen Anwendungsgebieten

Als Beispiel für eine anspruchsvolle Anwendung (zu der bisher noch keine befriedigenden Lösungen existieren) betrachten wir ein Informationssystem, das Angaben über die Korrosionsbeständigkeit von Werkstoffen enthält. Z.B. möchten wir folgende Aussage speichern:
„Der Stahl X2CrNiMo18 16 verliert in 5%iger $AlCl_3$-Lösung bei 323 K maximal 0.11mm/a und in 25%iger $AlCl_3$-Lösung bei 293 K 1.1 bis 11mm/a (15) und ist daher in Aluminiumchloridlösungen geringer Konzentration bei mäßig erhöhten Temperaturen beständig."
Eine Anfrage, zu der diese Aussage als Antwort ausgegeben werden müßte, könnte z.B. nach allen Werkstoffen suchen, die gegenüber Aluminiumchloridlösungen in einem bestimmten Temperatur- und Konzentrationsbereich beständig sind. Man muß also zunächst nach einer geeigneten Repräsentation für solche Sachverhalte suchen. Eine mögliche Darstellung des hier beteiligten Konzeptes „wässrige Aluminium-Chloridlösung"in Form eines semantischen Netzwerkes ist in Abbildung 1.2 dargestellt. Bislang ungelöst ist aber die Frage, wie IR-orientierte Retrievalmechanismen für derartige Repräsentationsformen aussehen sollten.

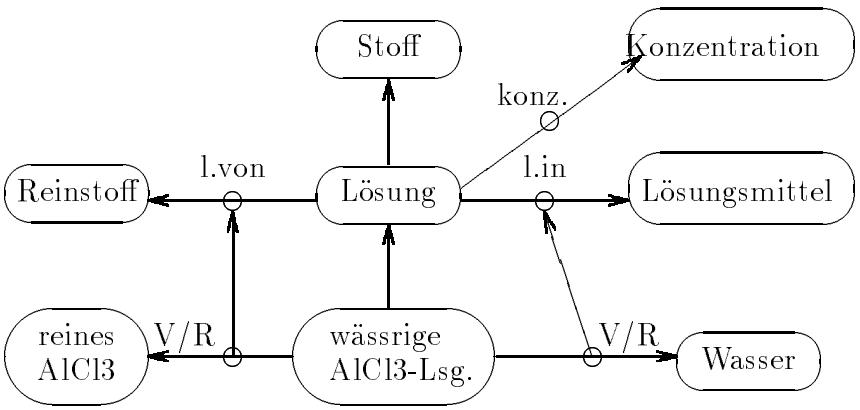


Abbildung 1.2: Semantisches Netz zu „wässrige $AlCl_3$-Lösung“

Kapitel 2

IR-Konzepte

2.1 Definition von Information Retrieval

Im Gegensatz zu anderen Teilgebieten der Informatik gibt es leider keine eindeutigen Definitionen für das Gebiet des IR. In [Rijsbergen 79a] findet sich der in Tabelle 2.1 dargestellte Vergleich von Information Retrieval mit Daten-Retrieval.

	Daten-Retrieval	*Information Retrieval*
Matching	exakt	partiell, best match
Inferenz	Deduktion	Induktion
Modell	deterministisch	probabilistisch
Klassifikation	monothetisch	polithetisch
Anfragesprache	formal	natürlich
Fragespezifikation	vollständig	unvollständig
gesuchte Objekte	die Fragespezifikation erfüllende	relevante
Reaktion auf Datenfehler	sensitiv	insensitiv

Tabelle 2.1: Daten-Retrieval vs. Information Retrieval

Eine aktuellere Definition des Gebietes findet sich in der Beschreibung der Aufgaben und Ziele der 1991 gegründeten Fachgruppe „Information Retrieval“ innerhalb der „Gesellschaft für Informatik“:
„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.“
Als kennzeichnend für das Gebiet werden somit vage Anfragen und unsicheres Wissen angesehen. Die Art der Darstellung des Wissens ist dabei von untergeordneter Bedeutung. Diese Auffassung liegt auch der Darstellung in diesem Skriptum zugrunde (nicht zuletzt deshalb, weil es gewisse Überlappungen zwischen den Verfassern der beiden Schriften gibt).

2.2 Daten — Wissen — Information

Datenbanksysteme enthalten Daten. IR-Systeme sollen die Suche nach Information[1] unterstützen. Enthalten IR-Systeme also Information? Schließlich ist vor allem in KI (Künstliche Intelligenz)-Publikationen häufig die Rede von Wissensbasen. Was ist denn nun der Unterschied zwischen Daten, Wissen und Information? In der deutschen Informationswissenschaft hat man sich vor einigen Jahren auf eine einheitliche Terminologie geeinigt, die in Abbildung 2.1 graphisch dargestellt ist. Danach sind Daten auf der syntaktischen Ebene anzusiedeln. In diesem Sinne wäre also eine Datenbasis eine nackte Sammlung von Werten ohne jegliche Semantik. Kommt Semantik hinzu, so sprechen wir von Wissen. Dementsprechend enthalten also Datenbanksysteme nicht nur Daten, sondern auch Wissen, weil zusätzlich zu den Daten zumindest ein Teil der Semantik des jeweiligen Anwendungsgebietes auch im System modelliert wird. Genauso enthält jedes IR-System Wissen (im Gegensatz etwa zu dem Fall, wo man Texte einfach in einer Datei abspeichert und mit Hilfe eines Texteditors durchsucht). Information schließlich ist auf der pragmatischen Ebene definiert. In [Kuhlen 90] wird

[1]Da Information keine exakt quantifizierbare Größe ist, gibt es auch den Plural „Informationen“ nicht. Es gibt nur mehr oder weniger Information.

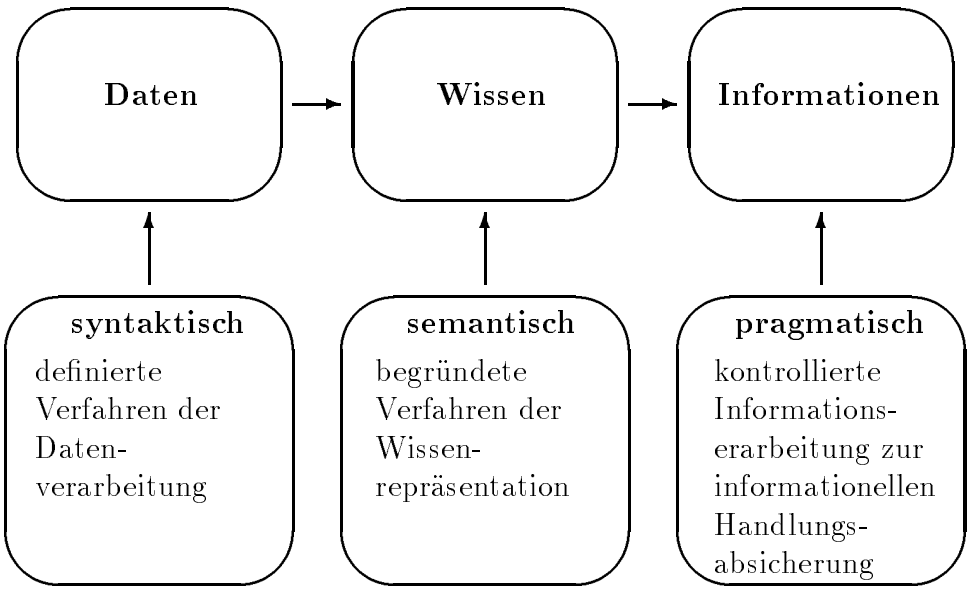


Abbildung 2.1: Daten — Wissen — Information

dies so formuliert: „Information ist die Teilmenge von Wissen, die von jemandem in einer konkreten Situation zur Lösung von Problemen benötigt wird“. Da diese Information häufig nicht vorhanden ist, wird danach in externen Quellen gesucht. Hierbei dient ein Informationssystem dazu, aus dem gespeicherten Wissen die benötigte Information zu extrahieren. Diese Transformation von Wissen in Information kann als ein Mehrwert erzeugender Prozeß angesehen werden [Kuhlen 91]. Ein wesentlicher Unterschied zwischen Information und Wissen besteht darin, daß Information flüchtig ist, Wissen dagegen permanent. Schlagwortartig läßt sich die Beziehung zwischen den beiden Begriffen ausdrücken durch die Formulierung „Information ist Wissen in Aktion“.

Als anschauliches Beispiel kann man hierzu die online verfügbaren UNIX-Manuals betrachten. Diese beinhalten Wissen über UNIX. Wenn nun ein Benutzer eines UNIX-Systems eine bestimmte Aktion ausführen möchte (z.B. ein Dokument drucken), aber nicht weiß, durch welche Kommandos er dies erreicht, so ist die in diesem Fall benötigte Information gerade die entsprechende Teilmenge des insgesamt in den Manuals verfügbaren, umfangreichen Wissens. Da nur ein geringer Teil des gesamten Wissens benötigt wird, besteht der Mehrwert der Information (so sie durch die hierzu verfügbaren Werkzeuge wie z.B. das man-Kommando geliefert wird) gerade in ihrer gezielten Bereitstellung.

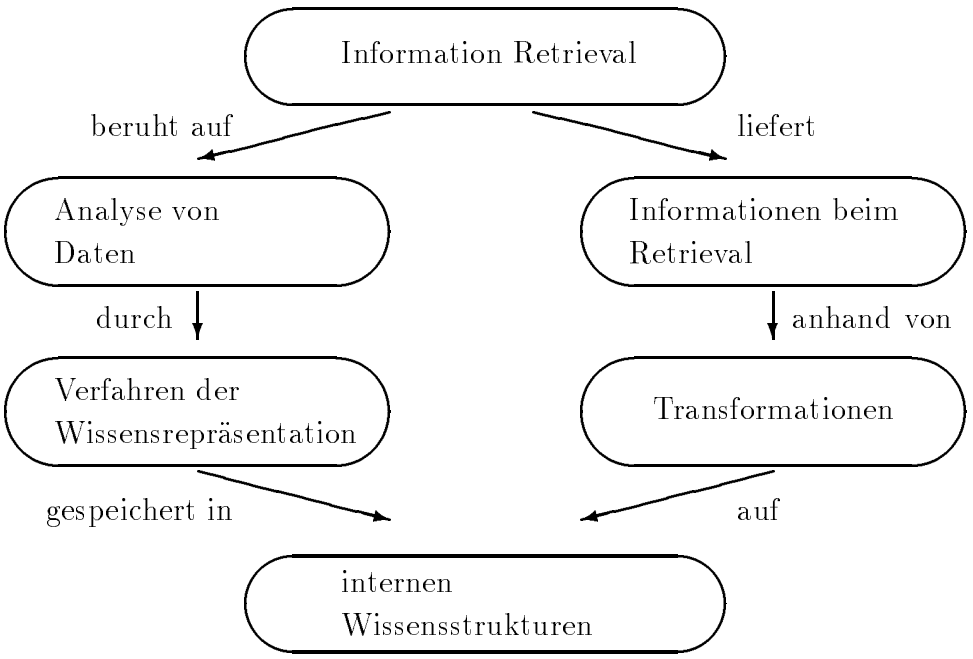


Abbildung 2.2: Grundmodell des Information Retrieval

2.3 Grundmodell des Information Retrieval

Wir stellen hier ein einfaches Grundmodell für Information Retrieval vor. Das allgemeine Modell ist in Abb. 2.2 dargestellt; die linke Seite der Abbildung repräsentiert dabei den Vorgang der Eingabe, während der Retrievalprozeß auf der rechten Seite dargestellt ist. Bei der Eingabe werden also Daten analysiert und dann in gespeichertes Wissen überführt. Beim Retrieval wird die benötigte Information durch Transformationen auf diesem gespeicherten Wissen erzeugt.
Die folgenden Abbildungen 2.3–2.5 zeigen die Spezialisierung dieses allgemeinen Modells für verschiedene Arten von IR-Systemen.

2.4 Konzeptionelles Modell für IR-Systeme

Wir beschreiben hier ein konzeptionelles Modell für Informationssysteme, das wir für die nachfolgenden Ausführungen zugrundelegen wollen. Dabei beschränken wir uns auf die Funktion der Informationssuche, während andere Aspekte solcher Systeme (z.B. die Aktualisierung der Datenbank oder zentrale vs. verteilte Datenhaltung) unberücksichtigt bleiben.
Das vorgeschlagene konzeptionelle Modell baut auf dem in [Meghini et al. 91] dargestellten Dokumentmodell auf, das wiederum eine Erweiterung des ODA-Dokumentmodells

Grundmodell des Information Retrieval, spezialisiert für

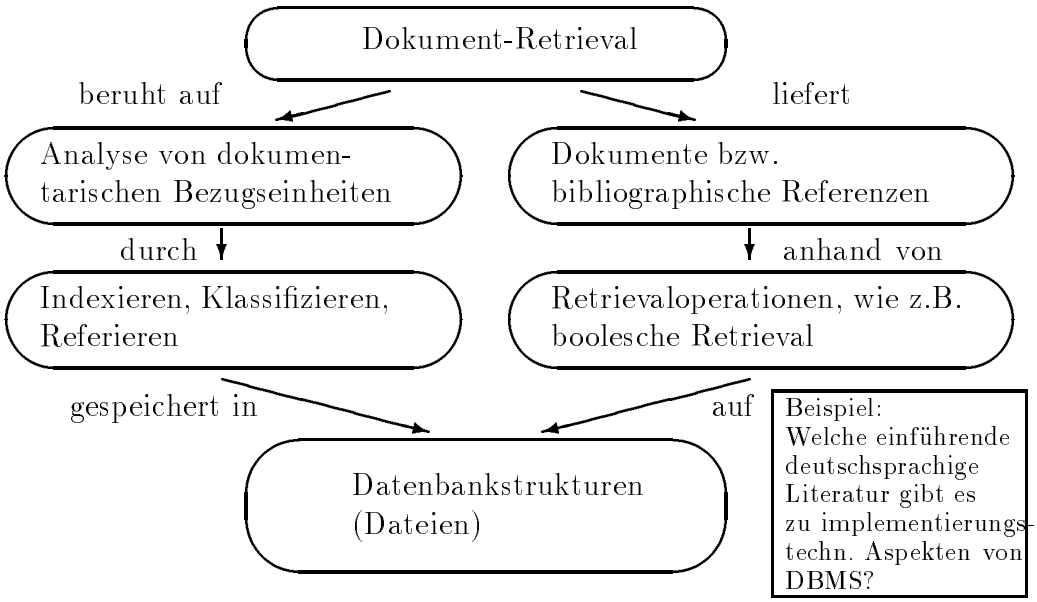


Abbildung 2.3: Grundmodell für Dokumentretrieval

Grundmodell des Information Retrieval, spezialisiert für

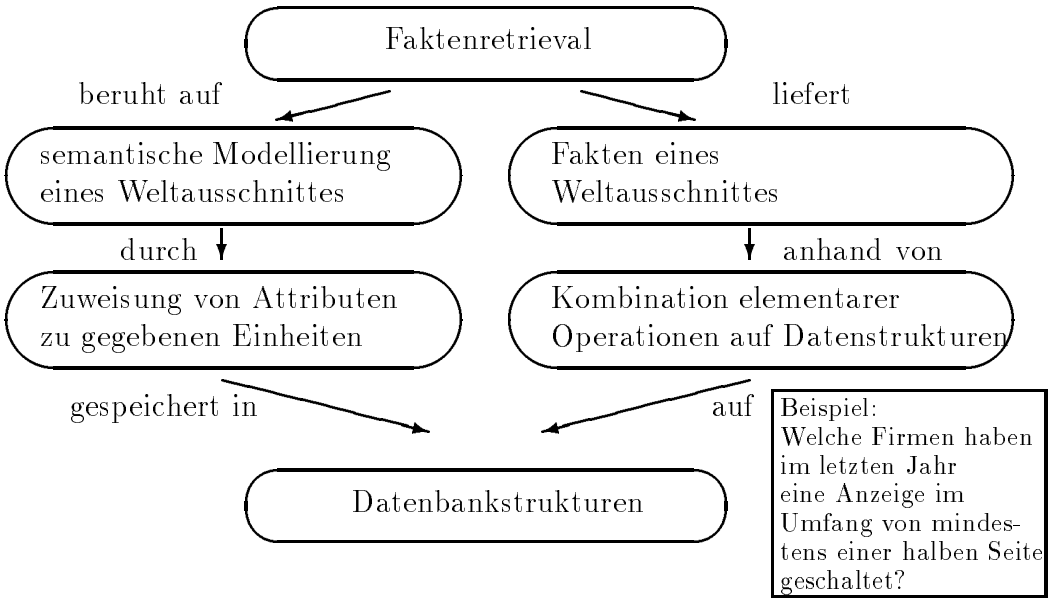


Abbildung 2.4: Grundmodell für Datenretrieval

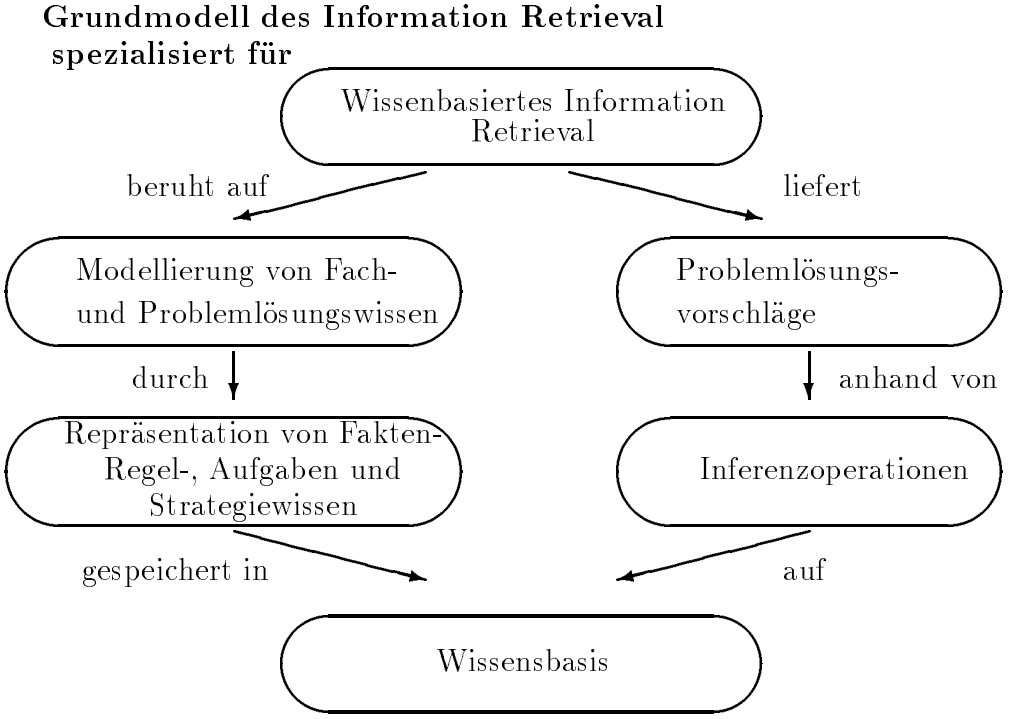


Abbildung 2.5: Grundmodell für wissensbasiertes IR

(s. 8.3.1) ist. Die wesentliche Idee dieses Dokumentmodells besteht darin, daß es mehrere Sichten auf den Inhalt eines Dokumentes gibt (siehe Abb. 2.6)

- Die Layout-Sicht beschreibt die Darstellung eines Dokumentes auf einem zweidimensionalen Medium.
- Die logische Sicht enthält die logische Struktur eines Dokumentes; diese umfaßt die zur Verarbeitung (z.B. Editieren) notwendigen Informationen, also im wesentlichen den Inhalt ohne die Layout-Struktur.
- Die semantische Sicht bezieht sich auf die Semantik des Inhalts eines Dokumentes. Diese Sicht ist im ODA-Standard nicht enthalten; für IR-Systeme ist eine solche Sicht aber essentiell, da ansonsten nur primitive Suchoperationen in der Form der Zeichenkettensuche möglich sind.

Prinzipiell lassen sich diese drei Sichten auf beliebige Objekte in Datenbanken anwenden. Bei herkömmlichen Datenbanken wird nur die logische Sicht unterstützt, da dies für die meisten Anwendungen völlig ausreicht. Solange an die Darstellung der Objekte keine besonderen Anforderungen gestellt werden, reichen meist die generischen Ausgabeformate der interaktiven Anfrageschnittstelle aus. Darüber hinaus werden aber für Standard-Anwendungen spezielle Ausgabemasken erstellt, so daß diese im Prinzip die Layout-Sicht realisieren (allerdings ist die Verwaltung dieser Masken nicht in das Datenbanksystem integriert). Für eine zusätzliche semantische Sicht bestand bislang bei Datenbanksystemen keine Notwendigkeit, da diese bei den vorherrschenden

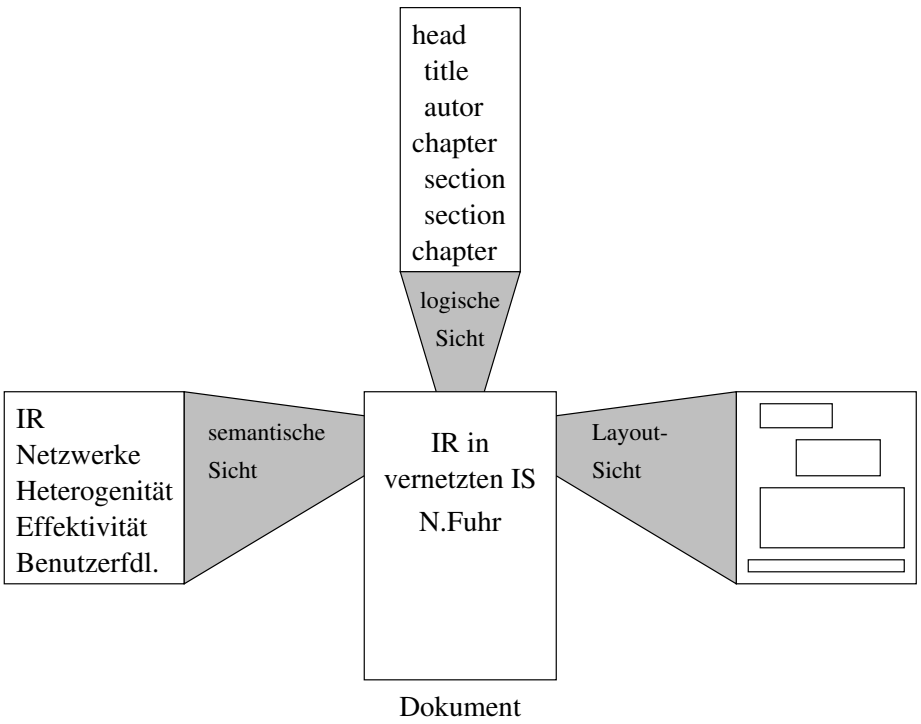


Abbildung 2.6: Sichten auf Dokumente

kaufmännischen und administrativen Anwendungen mit der logischen Sicht identisch ist.

Aufbauend auf diesen drei Sichten auf Dokumente und Datenbank-Objekte im allgemeinen läßt sich das in Abbildung 2.7 dargestellte konzeptionelle Modell formulieren. In einem Informationssystem werden Objekte aus der zu modellierenden Anwendung („reales Objekt“) in einer internen Repräsentation in der Datenbank gespeichert, die wir hier als DB-Objekt bezeichnen. Ein solches Objekt kann z.B. ein Textdokument oder eine Menge von Fakten sein, die das reale Objekt beschreiben; im ersteren Falle würde die interne Repräsentation etwa der Form eines Textes entsprechen, wie er z.B. mit einem Texteditor erstellt wurde.

Aus dieser Repräsentation wird sowohl die logische wie auch die semantische und die Layout-Sicht des Objektes abgeleitet. Zum Beispiel wird bei vielen Forschungsansätzen im Bereich des IR für Texte folgende semantische Sicht zugrundegelegt: Der Text besteht aus einzelnen Abschnitten; diese Textabschnitte werden in Worte zerlegt, die wiederum auf Grund- oder Stammform reduziert werden; für diese Terme bestimmt man dann die Vorkommenshäufigkeit im Text und in der gesamten Datenbasis. Bei herkömmlichen kommerziellen IRS werden dagegen die Textabschnitte nur als Folge von Wörtern aufgefaßt, wobei ein Wort eine durch Leer- oder Interpunktionszeichen begrenzte Buchstabenfolge ist. Somit existiert hier keine eigentliche

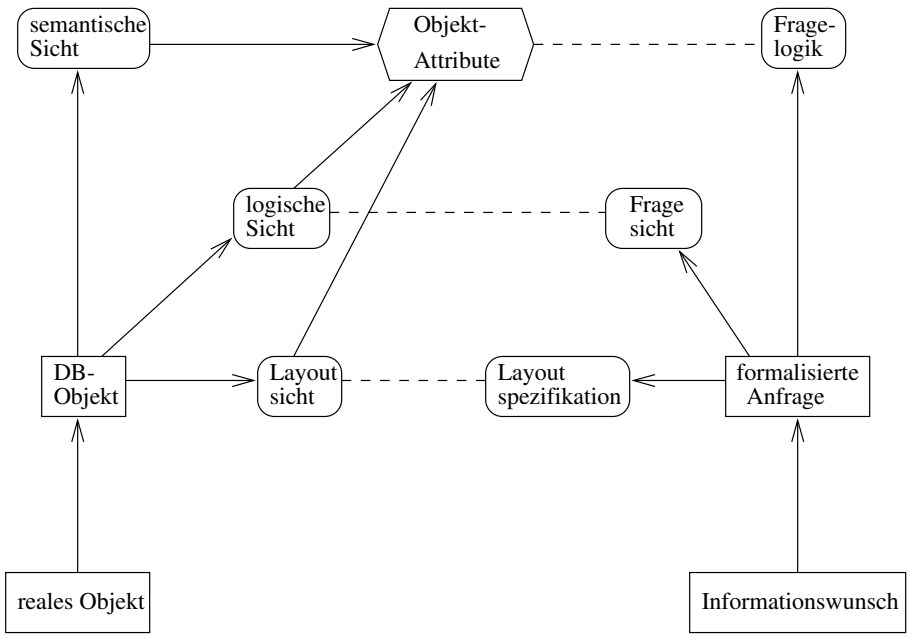


Abbildung 2.7: Konzeptionelles Modell für Informationssysteme

semantische Sicht, sondern es wird nur mit der logischen Sicht gearbeitet.

Vom Standpunkt des IR interessiert besonders, welche Arten von Anfragen bezüglich eines Objektes möglich sind. Dieser Aspekt wird im konzeptionellen Modell durch die Objektattribute dargestellt; Anfragen können sich nur auf diese Attribute beziehen. Die Attribute wiederum können sich aus allen drei Sichten ableiten. Dadurch wird es möglich, sowohl Fragen nach der Semantik von Objekten zu stellen als auch Objekte mit bestimmter logischer oder Layout-Struktur zu suchen. Beispiele für Anfragen mit Bezug zu den verschiedenen Arten von Attributen wären in einem Büro-Informationssystem etwa „Suche alle Informationen über Büromöbel“ (semantisch), „Suche alle Rechnungen der Firma Meier“ (logisch) und „Suche einen Brief, der ein blaues Logo in der rechten oberen Ecke enthält“ (Layout). In der Regel werden in einer Anfrage aber Bedingungen an die verschiedenen Arten der Attribute in Kombination auftreten.

Auf der Seite der Fragen ergibt sich eine ähnliche Struktur wie bei den Objekten. Ein Informationswunsch eines Benutzers muß dem System gemäß der verwendeten Anfragesprache als formalisierte Anfrage übergeben werden. Bei kommerziellen IRS ist dies z.B. häufig ein boolescher Ausdruck oder eine Folge von ausgewählten Menüpunkten, bei experimentellen Systemen werden meist natürlichsprachliche Formulierungen zugelassen. Prinzipiell umfaßt die formalisierte Anfrage alle Eingaben eines Benutzers während der Bearbeitung einer Anfrage, also z.B. auch Relevanzbeurteilungen zu einzelnen ausgegebenen Dokumenten. Aus der formalisierten Anfrage wird dreierlei

abgeleitet:

- Die Fragelogik enthält die Bedingungen an die Objektattribute, die ein Antwortobjekt erfüllen sollte. Wie oben erläutert, können sich diese Bedingungen aber nur auf die Objektattribute beziehen; auf Aspekte eines Objektes, die nicht durch entsprechende Attribute repräsentiert sind, kann auch nicht in der logischen Frageformulierung bezug genommen werden. Diese Tatsache ist besonders wichtig bei multimedialen Informationssystemen, die zwar häufig die Speicherung von Audio- und Videosequenzen erlauben, aber nur sehr beschränkte Möglichkeiten zur Suche nach solchen Objekten anbieten.
- Die Frage-Sicht definiert, welche Teile des Inhalts eines gefundenen Objektes als Antwort ausgegeben werden sollen. In relationalen Datenbanksystemen wird diese Sicht z.B. mit Hilfe des Projektions-Operators definiert. Für komplexer strukturierte Objekte (wie sie z.B. in objektorientierten Datenbanken zu finden sind) müssen hier natürlich auch entsprechend mächtigere Operationen angeboten werden. Bei der inhaltlichen Suche nach Texten oder multimedialen Dokumenten wäre es wünschenswert, wenn nur die Teile des Objektes, die für die Beantwortung der Frage relevant sind, angezeigt würden. Dies setzt natürlich eine tiefergehende Inhaltserschließung voraus.
- Die Layout-Spezifikation schließlich legt fest, in welcher Form die Sicht auf ein oder mehrere Antwortobjekte dargestellt werden soll.

Anhand dieser Abbildung kann auch der Aspekt der Unsicherheit verdeutlicht werden. Bei vielen technisch-wissenschaftlichen Anwendungen ist schon die Abbildung eines realen Objektes in die Objektrepräsentation mit Unsicherheit behaftet, weil dessen Eigenschaften (etwa Meßwerte) nur mit einer begrenzten Genauigkeit oder nur unvollständig erhoben werden können. Die Ableitung der Objektsemantik aus der internen Repräsentation ist eine weitere Quelle von Unsicherheit. Speziell bei Texten oder multimedialen Dokumenten kann deren Inhalt nur unzureichend erschlossen werden, und durch die anschließende Verdichtung in Form der Objektattribute ergibt sich ein weiterer Informationsverlust.

Auf der Seite der Fragen ergeben sich die gleichen Probleme der Unsicherheit, insbesondere bei der Abbildung auf die interne Repräsentation. Zusätzlich spielt hier das für IR-Anwendungen typische Moment der Vagheit eine wichtige Rolle (Im Prinzip wären auch auf der Seite der Objekte vage Beschreibungen möglich, was wir aber hier nicht weiter betrachten wollen). Daher sollte die Frageformulierung in der Lage sein, diese Vagheit zu repräsentieren. Bei probabilistischen Textretrievalsystemen geschieht dies z.B. durch eine Gewichtung der Frageterme.

Kapitel 3

Evaluierung

Wie in kaum einem anderen Teilgebiet der Informatik spielt die Evaluierung von Verfahren im Information Retrieval eine wichtige Rolle. Aufgrund der Komplexität der Aufgabenstellung sind nicht-experimentelle Methoden zur Beurteilung von Retrievalverfahren kaum geeignet. Zudem ist die Forschungsliteratur im IR voll von Beispielen von plausibel und mächtig erscheinenden Verfahren, die entweder gar nicht praktisch umsetzbar waren oder aber bezüglich der erreichten Retrievalqualität bei weitem nicht an einfachere, aber wirkungsvollere Verfahren heranreichten.

3.1 Effizienz und Effektivität

Wenn man von Bewertung von IR-Methoden spricht, so muß man zwischen Effizienz und Effektivität unterscheiden. Unter Effizienz versteht man den möglichst sparsamen Umgang mit Systemressourcen für eine bestimmte Aufgabe. Zu diesen Ressourcen zählen hauptsächlich:

- Speicherplatz,
- CPU-Zeit,
- Anzahl I/O-Operationen,
- Antwortzeiten.

In den übrigen Gebieten der Informatik kann man sich meist auf reine Effizienzbetrachtungen beschränken, weil dort die vom System zu lösenden Aufgabenstellungen klar definiert sind und eine korrekte und vollständige Lösung der Aufgaben durch das System unabdingbare Voraussetzung ist. Im Information Retrieval dagegen muß man akzeptieren, daß es praktisch kein System gibt, das die hier betrachteten Aufgabenstellungen perfekt löst. Ein wesentlicher Unterschied zwischen einzelnen Systemen besteht gerade in der Qualität, mit der ein System die gewünschten Leistungen erbringt.

Effektivität bezeichnet das Kosten-Nutzen-Verhältnis bei der Anwendung eines bestimmten Verfahrens. Bei der Nutzung eines IR-System bestehen die „Kosten“ in dem vom Benutzer aufzubringenden Zeitaufwand und seiner mentalen Belastung bei der Lösung seines Problems mithilfe des Systems. Der erzielte Nutzen besteht in der Qualität der erreichten Lösung. (Ein einfaches Beispiel hierfür wäre z.B. die Suche eines Studenten nach Literatur zur Prüfungsvorbereitung in einem bestimmten Fach; der erzielte Nutzen könnte dann z.B. an der erreichten Note gemessen werden). Wir werden die folgenden Betrachtungen allein auf die resultierende Qualität eines Informationssystems beschränken — nicht zuletzt deshalb, weil die hier betrachteten Verfahren keine oder nur wenige, standardisierte Interaktionsmöglichkeiten zwischen Benutzer und System zulassen.

3.2 Relevanz

Um die Qualität der Antworten eines IR-Systems zu beurteilen, legt man meist das Konzept der Relevanz zugrunde: Relevanz bezeichnet dabei eine Eigenschaft der Beziehung zwischen der Anfrage und einem einzelnen Element der Antwortmenge. Hierbei werden folgende Annahmen gemacht:

- Die Systemantwort ist eine Menge von Objekten (z.B. Dokumente). Damit werden stärker strukturierte Antworten nicht berücksichtigt. Wie unten gezeigt wird, lassen sich die hier diskutierten Evaluierungsmethoden aber leicht auf lineare Anordnungen (Rangordnungen) ausdehnen.
- Die Qualität des Objekts, also seine Relevanz bezüglich der Anfrage, hängt nur von der Anfrage ab. Wechselseitige Abhängigkeiten zwischen Objekten bleiben dagegen unberücksichtigt (wenn z.B. die Bedeutung eines bestimmten Dokumentes erst nach der Lektüre eines anderen Dokumentes erkannt wird).

Ebenso unberücksichtigt bleibt die Tatsache, daß die Beziehung zwischen Informationsbedürfnis und Anfrage relativ komplex sein kann und sich nur schlecht auf eine lineare Skala abbilden läßt.
Im folgenden machen wir zudem die Einschränkung, daß die Relevanzskala zweistufig ist, also aus den beiden Werten „relevant“ und „nicht relevant“ besteht.

3.3 Distributionen

Distributionen sind abstrakte Darstellung von Retrievalantworten, die als Grundlage für Bewertungsmaße dienen. Wir illustrieren dieses Konzept anhand eines Beispiels: Als Antwort auf eine Anfrage berechne ein System folgende Retrievalwerte für die Dokumente in der Datenbasis:

$$\{(d_1, 0.3), (d_2, 0.8), (d_3, 0.1), (d_4, 0.8), (d_5, 0.8), (d_6, 0.6), (d_7, 0.3), (d_8, 0.1)\}$$

Daraus ergibt sich folgende Rangordnung bzw. **Distribution von Dokumenten**:

$$(\{d_2, d_4, d_5\}, \{d_6\}, \{d_1, d_7\}, \{d_3, d_8\})$$

Die Relevanzbeurteilung des Benutzers sei nun folgende (R — relevant, $\bar{R}$ — nicht relevant):

$$\{(d_1, R), (d_2, R), (d_3, \bar{R}), (d_4, R), (d_5, R), (d_6, \bar{R}), (d_7, R), (d_8, R)\}$$

Durch die Zusammenführung von Rangordnung un Relevanzurteilen erhält man die **Distribution mit Relevanzurteilen**

$$(\{d_2^+, d_4^+, d_5^+\}, \{d_6^-\}, \{d_1^+, d_7^+\}, \{d_3^-, d_8^+\}).$$

Für die Bewertung der Retrievalqualität abstrahiert man nun von spezifischen Dokumenten. Dadurch ergeben sich Äquivalenzklassen von Distributionen mit Relevanzurteilen, die wir im folgenden einfach als **Distributionen** bezeichnen:

$$\Delta = (+ + +| - | + +| + -)$$

3.4 Standpunkte und Bewertungsmaße

Jedem Bewertungsmaß liegt ein bestimmter Standpunkt bzgl. des „Besserseins“ einer Distribution im Vergleich zu einer anderen zugrunde. Bevor man ein Maß anwendet, sollte man sich daher im klaren darüber sein, welcher Standpunkt dem gewählten Maß zugrundeliegt und ob dieser für die aktuelle Anwendung adäquat ist.

3.4.1 Benutzerstandpunkte

Wir nehmen an, daß das IRS als Antwort auf eine Anfrage eine Rangordnung von Dokumenten produziert, die der Benutzer sequentiell solange durchsieht, bis ein bestimmtes Abbruchkriterium erfüllt ist. Für jedes Kriterium (= Standpunkt) kann man dann ein entsprechendes Bewertungsmaß definieren, das die Präferenzen des Benutzers widerspiegelt. Beispiele für mögliche Abbruchkriterien und zugehörige Bewertungsmaße sind:

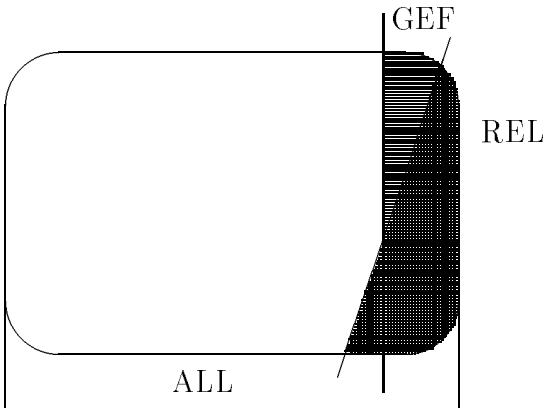
- n Dokumente gesehen: # gesehene relevante Dokumente
- n relevante Dokumente gesehen: # gesehene Dokumente
- n nicht relevante Dokumente gesehen: # gesehene / # gesehene relevante Dokumente
- n nicht relevante Dokumente in Folge gesehen: # gesehene / # gesehene relevante Dokumente

3.4.2 Benutzer- vs. Systemstandpunkte

Man kann grob zwischen Benutzer- und Systemstandpunkten unterscheiden. Erstere spiegeln dabei die Sicht eines einzelnen Benutzers wider, während letzteren ein globale Sicht (die des Systembetreibers) zugrundeliegt. Dementsprechend beziehen sich **benutzerorientierte Maße** auf das mögliche Verhalten und die Präferenzen der Benutzer. **Systemorientierte Maße** entsprechen dagegen einer systemorientierten Sicht, die unabhängig von speziellen Benutzerstandpunkten ist. Daher wird eine „globale" Bewertung der Distribution angestrebt. Im Gegensatz dazu werden etwa bei den obigen benutzerorientierten Maßen jeweils nur die ersten Dokumente der Rangordnung betrachtet. Ein einfaches systemorientiertes Maß wäre daher die Korrelation zwischen Systemantwort Δ und idealer Distribution $\bar{\Delta}$, z.B. $\Delta = (+++|-|++|+-)$ und $\bar{\Delta} = (+++++++|--)$. Als Beispiel für ein systemorientiertes Maß wird in 3.9 das Nützlichkeitsmaß vorgestellt.

3.5 Maße für boolesches Retrieval

Wir betrachten zunächst den Fall der Retrievalbewertung für boolesches Retrieval, da die Maße für Rangordnungen Erweiterungen der Maße für boolesches Retrieval sind. Als Benutzerstandpunkt wird hier angenommen, daß der Benutzer sich stets alle gefundenen Dokumente anschaut. Im folgenden bezeichne GEF die Menge der gefundenen Antwortobjekte, REL die Menge der relevanten Objekte in der Datenbank und ALL die Gesamtzahl der Dokumente in der Datenbank.



Basierend auf diesen Mengen lassen sich dann die Maße Precision, Recall und Fallout wie folgt definieren:

$$\text{Precision:} \quad p = \frac{|REL \cap GEF|}{|GEF|}$$

$$\text{Recall:} \quad r = \frac{|REL \cap GEF|}{|REL|}$$

$$\text{Fallout:} \quad f = \frac{|GEF - REL|}{|ALL - REL|}$$

Hierbei gibt Precision den Anteil der relevanten an den gefundenen Dokumenten wieder. Recall dagegen bezeichnet den Anteil der relevanten Dokumente, die tatsächlich gefunden wurden. Schließlich mißt Fallout den Anteil der gefundenen irrelevanten an allen irrelevanten Dokumenten der Kollektion; hiermit wird also die Fähigeit des Systems bewertet, irrelevante Dokumente vom Benutzer fernzuhalten.
Die Größe der Precision ist für jeden Benutzer eines IR-Systems direkt ersichtlich. Die Größe des Recall ist dagegen für einen Benutzer weder erkennbar, noch kann sie mit vernünftigem Aufwand präzise bestimmt werden. Der Grund hierfür liegt in dem Problem, die Mächtigkeit der Menge REL präzise zu bestimmen. Folgende Näherungsmethoden wurden hierzu vorgeschlagen:

1. Vollständige Relevanzbeurteilung einer repräsentativen Stichprobe der gesamten Datenbasis: Da REL sehr viel kleiner als die gesamte Datenbasis ist (z.B. mögen 100 von 10^6 Dokumenten relevant sein), müßte die repräsentative Stichprobe schon einen relativ großen Teil der Datenbasis umfassen, was zuviel Beurteilungsaufwand erfordert.
2. Dokument-Source-Methode: Hierbei wählt man ein zufälliges Dokument aus der Datenbank und formuliert dann eine Frage, auf die dieses Dokument relevant ist. Anschließend wird geprüft, ob das System das betreffende Dokument als Antwort auf die Frage liefert. Für eine Menge von Fragen schätzt man dann über die relative Häufigkeit die Wahrscheinlichkeit, daß das Source-Dokument gefunden wird, als Näherung des Recalls. Nachteil dieser Methode ist, daß die verwendeten Fragen keine echten Benutzerfragen sind.
3. Frageerweiterung: Man erweitert die ursprünglichen Anfrage, so daß eine Obermenge der ursprünglichen Antwortmenge gefunden wird, die wesentlich größer ist und weitere relevante Dokumente enthält (z.B. kann man auch mehrere Frageformulierungen von verschiedenen Bearbeitern erstellen lassen und die Vereinigungsmenge der Antwortmengen betrachten). Damit erhält man aber nur eine Teilmenge der Menge REL, somit sind die darauf basierenden Recall-Schätzungen im allgemeinen zu hoch.
4. Abgleich mit externen Quellen: Man versucht parallel zur Datenbanksuche noch mit davon unabhängigen Methoden, relevante Dokumente zu bestimmen (z.B. indem man den Fragenden oder andere Fachleute fragt, 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. Nachteile dieser Methode sind, daß sie zum einen recht aufwendig ist, zum

anderen oft nicht anwendbar ist, weil es keine unabhängigen externen Quellen gibt.

Hat man für eine Frage Recall und Precision bestimmt, so läßt sich dieses Ergebnis als Punkt in einem Recall-Precision-Graphen darstellen. Beim Vergleich zweier Systeme bezüglich einer Frage ist dann dasjenige System besser, das sowohl einen höheren Recall- als auch einen besseren Precision-Wert liefert (einer der beiden Werte darf auch gleich sein). In Abbildung 3.1 sind die Bereiche, in denen bessere bzw. schlechtere Ergebnisse liegen, weiß markiert. Häufig wird allerdings ein System einen höheren Recall, das andere dagegen eine höhere Precision liefern, so daß sich keine Aussage bezüglich einer Überlegenheit eines der beiden Systeme ableiten läßt (die grauen Bereiche in Abbildung 3.1).

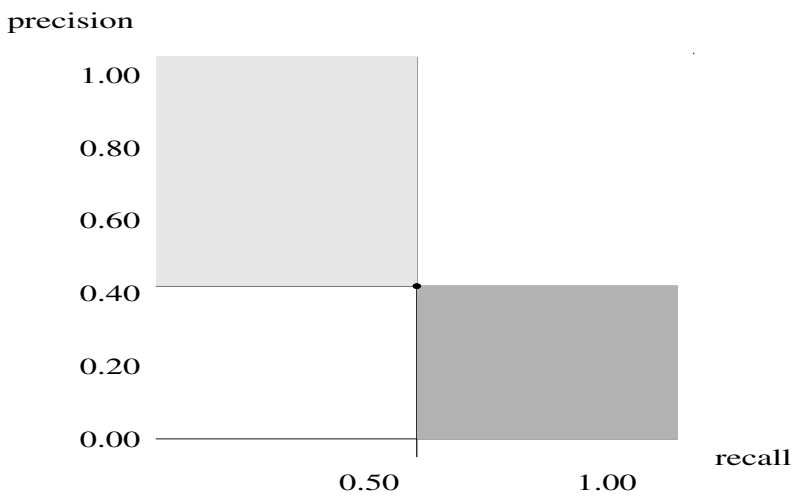


Abbildung 3.1: Darstellung eines Retrievalergebnisses als Punkt im Recall-Precision-Graphen

3.5.1 Mittelwertbildung

Da es sich bei Retrievalexperimenten um stochastische Experimente handelt, muß man eine Menge von Fragen betrachten, um fundierte Aussagen über die Qualität eines Systems zu erhalten. Dementsprechend müssen Mittelwerte für die Qualitätsmaße berechnet werden. Hierzu werden im IR zwei verschiedene Methoden angewendet (im folgenden gehen wir von N Fragen aus, wobei REL_i und GEF_i für $i = 1, \ldots, N$ die jeweiligen Mengen gefundener bzw. relevanter Dokumente bezeichnen):

- Bei der **Makrobewertung** wird das arithmetische Mittel der Werte für die

einzelnen Fragen gebildet, also z.B. für die Precision:

$$p_M = \frac{1}{N}\sum_{i=1}^{N}\frac{|REL_i \cap GEF_i|}{|GEF_i|} \tag{3.1}$$

Probleme ergeben sich bei der Makrobewertung, wenn einzelne Fragen leere Antwortmengen liefern (dies ist z.B. häufig bei Tests der Fall, wo nur eine Stichprobe der Dokumente der gesamten Datenbasis verwendet wird, so daß Fragen mit wenigen Antworten auf der gesamten Datenbasis oft keine Antwort in der Stichprobe liefern). Durch verbesserte probabilistische Schätzmethoden kann diese Problem u.U. behoben werden (siehe 6.5.1).
Bei der Makrobewertung geht jede Frage gleich stark in den Mittelwert ein, was nicht immer wünschenswert sein mag (wenn man Fragen mit größeren Antwortmengen stärker gewichten will). Daher bezeichnet man diese Methode auch als Frage- oder Benutzer-orientiert.

- Bei der **Mikrobewertung** werden zuerst Zähler und Nenner des Maßes addiert, bevor der Quotient gebildet wird — also bei der Precision:

$$p_\mu = \frac{\sum_{i=1}^{N}|REL_i \cap GEF_i|}{\sum_{i=1}^{N}|GEF_i|} \tag{3.2}$$

Dadurch wird das Problem der leeren Antwortmengen umgangen. Da hier jedes Dokument gleichstark in den Mittelwert eingeht, bezeichnet man die Mikrobewertung auch als Dokument- oder System-orientiert.

Ein wesentliches Problem der Mikrobewertung ist aber die fehlende Monotonieeigenschaft: Wir betrachten zwei verschiedene Retrievalergebnisse Δ_1, Δ_2, die von zwei Systemen zur gleichen Frage geliefert worden sind. Ein Maß ist dann monoton, wenn sich durch das Hinzufügen des gleichen Retrievalergebnisses Δ zu beiden Ergebnissen die Aussage über die Überlegenheit eines der beiden Systeme nicht ändert.

$$\begin{aligned} \text{Sei } p_\mu(\Delta_1) = \frac{1}{2}, \quad p_\mu(\Delta_2) = \frac{2}{5} \qquad & \text{und } p_\mu(\Delta) = \frac{2}{8} \\ \text{Dann ist } p_\mu(\Delta_1) = \frac{1}{2} \quad > \quad & \frac{2}{5} = p_\mu(\Delta_2), \text{ aber} \\ p_\mu(\Delta_1, \Delta) = \frac{3}{10} \quad < \quad & \frac{4}{13} = p_\mu(\Delta_2, \Delta). \end{aligned}$$

3.6 Rangordnungen

Mit Ausnahme des booleschen Retrieval liefern alle Retrievalverfahren eine Rangordnung von Dokumenten als Antwort. Daher müssen die Definitionen der Retrievalmaße entsprechend erweitert werden.

n	Dokumentnr.	x=rel.	Recall	Precision
1	588	x	0.2	1.00
2	589	x	0.4	1.00
3	576		0.4	0.67
4	590	x	0.6	0.75
5	986		0.6	0.60
6	592	x	0.8	0.67
7	984		0.8	0.57
8	988		0.8	0.50
9	578		0.8	0.44
10	985		0.8	0.40
11	103		0.8	0.36
12	591		0.8	0.33
13	772	x	1.0	0.38
14	990		1.0	0.36

Tabelle 3.1: Recall-Precision nach n Dokumenten bei linearer Ordnung

Bei Rangordnungen muß man zusätzlich unterscheiden, ob eine lineare (totale) Ordnung der Dokumente aus der Datenbasis vorliegt oder nur eine schwache Ordnung (d.h., es können mehrere Dokumente im selben Rang sein).
Wir stellen Retrievalergebnisse meist durch folgendes Schema dar:

$$\Delta_1 = (+--|+++-------) \tag{3.3}$$

Die einzelnen Ränge werden dabei durch „|“ getrennt, „+“ bezeichnet ein relevantes und „-“ ein nichtrelevantes Dokument. Bei Δ_1 handelt es sich um eine schwache Ordnung. Lineare Ordnungen sind dagegen z.B.

$$\Delta_2 = (+|+|-|+|-|+|-|-|-|-|-|+|-). \tag{3.4}$$
$$\Delta_3 = (+|-|+|+|+|-|-|-|-|-|-|-|+|-|+|-) \tag{3.5}$$

3.6.1 Lineare Ordnung

Bei einer linearen Ordnung können Recall und Precision (r, p) für eine Anfrage in Abhängigkeit von der Mächtigkeit in der Antwortmenge bestimmt werden, wie dies am Beispiel in Tabelle 3.1 gezeigt wird. Δ_2 (3.4) ist die zugehörige Darstellung des Retrievalergebnisses.
Trägt man die sich für verschiedene n ergebenden (r, p)-Werte in das Recall-Precision-Diagramm ein, so ergibt sich das in Abbildung 3.2 gezeigte Bild. Um die Übersichtlichkeit zu erhöhen, kann man die einzelnen Punkte mit Geradenstücken verbinden (siehe

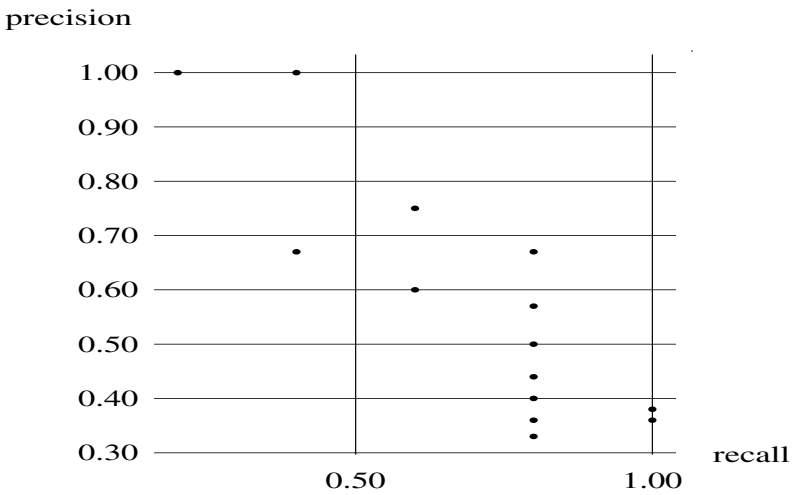


Abbildung 3.2: Graphische Darstellung der Werte aus Tabelle 3.1

Abb. 3.3). Allerdings darf man den Punkten auf diesen Geradenstücken keine Bedeutung zuordnen (um z.B. Zwischenwerte zu interpolieren)! Diese Art der Darstellung ist besonders nützlich, wenn man die die Qualitätsmaße für mehrere Rangordnungen in einem einzigen Graphen darstellen möchte (siehe Abb. 3.4 und 3.5).
Um die Kurven im R-P-Graphen interpretieren zu können, wird in [Salton & McGill 83, S. 167-8] vorgeschlagen, die Originalkurve wie in Abb. 3.6 dargestellt zu interpolieren. Dabei wird jeder einzelne (r, p) Wert durch eine waagerechte Linie bis zu $r = 0$ interpoliert. Der resultierende Graph ergibt sich dann als das Maximum über diese Geradenstücke.

3.6.2 Schwache Ordnung

Viele Retrievalverfahren liefern nur eine schwache Ordnung auf der Antwortmenge, also mit mehreren Dokumenten in einem Rang (wie z.B. der "coordination level match" oder als Extremfall das boolesche Retrieval). In diesem Fall darf man nicht versuchen, durch zufällige Anordnung der Dokumente eines Ranges eine lineare Ordnung zu erzeugen, um dann die dazu passenden Evaluierungsmethoden anzuwenden; wie in [Raghavan et al. 89] dargestellt, ergeben sich dadurch unzuverlässige, nicht reproduzierbare Ergebnisse.
Tabelle 3.2 zeigt ein solches Retrievalergebnis und Tabelle 3.3 die zugehörigen (r, p)-Werte für die einzelnen Ränge, die in Abb. 3.7 graphisch dargestellt sind.
Ein möglicher Weg zur Definition von eindeutigen Precision-Werten für vorgegebene Recall-Werte besteht in der sogenanten PRECALL-Methode. Diese Methode ist an Salton's Interpolationsmethode angelehnt. Hierbei wird zunächst die Precision an

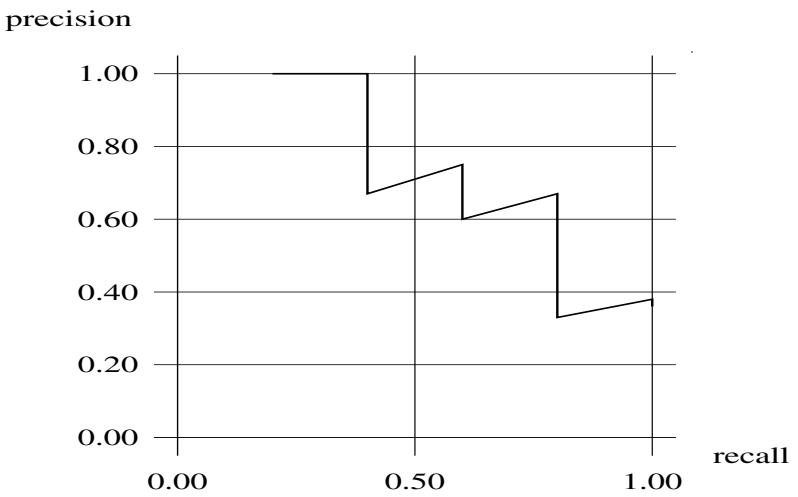


Abbildung 3.3: Darstellung der Werte aus Tabelle 3.1 als Kurve

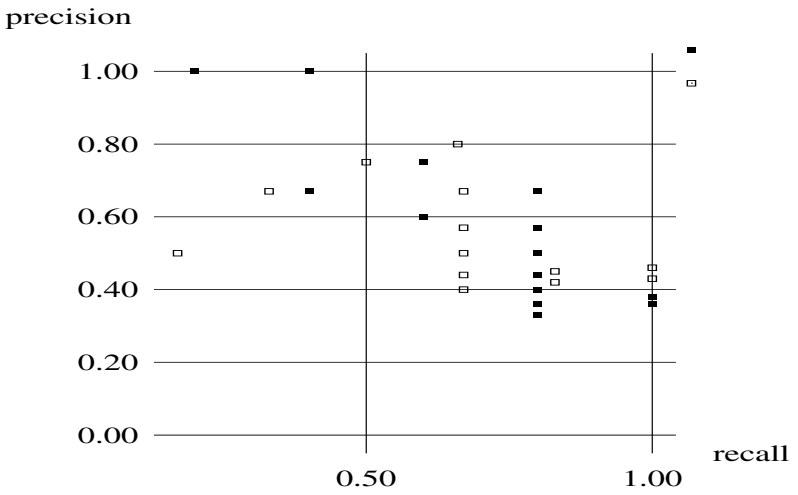


Abbildung 3.4: Graphische Darstellung der Werte für zwei verschiedene Rangordnungen (Δ_2 (3.4) und Δ_3 (3.5))

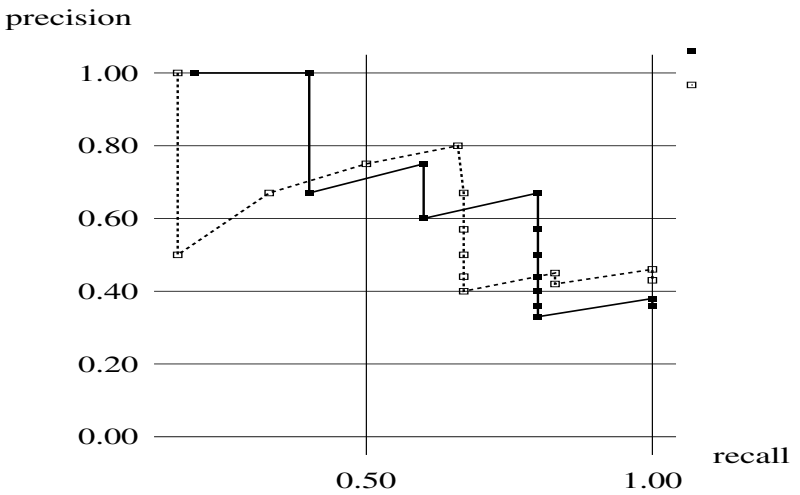


Abbildung 3.5: Darstellung der Werte aus Abb. 3.4 als Kurve

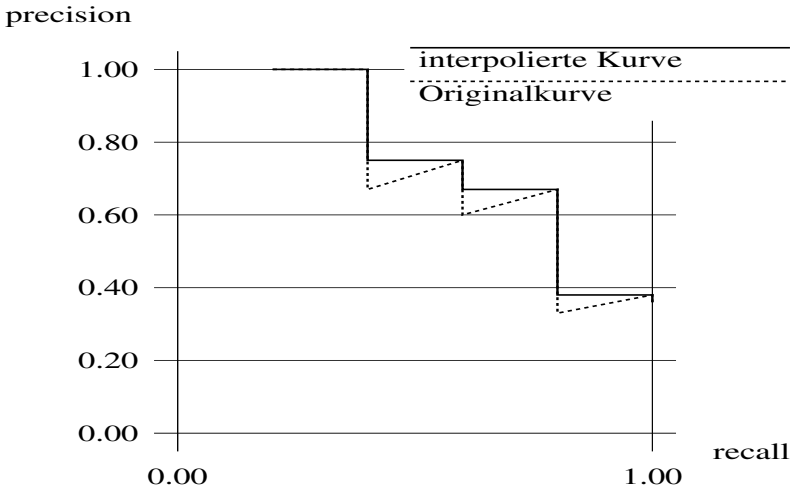


Abbildung 3.6: R-P-Graph nach Salton

Rang	Alg. 1		Alg. 2	
	rel.	nicht rel.	rel.	nicht rel.
1	0	1	0	1
2	2	7	6	9
3	4	0	2	0
4	4	2	2	0
5	0	80	0	80
	10	90	10	90

Tabelle 3.2: Beispiel für schwache Ordnung

	Algorithmus 1		Algorithmus 2	
Rang	Recall	Precision	Recall	Precision
1	0.00	0.00	0.00	0.00
2	0.20	0.20	0.60	0.38
3	0.60	0.43	0.80	0.44
4	1.00	0.50	1.00	0.50
5	1.00	0.10	1.00	0.10

Tabelle 3.3: Rangabhängige Recall und Precision zu Tab. 3.2

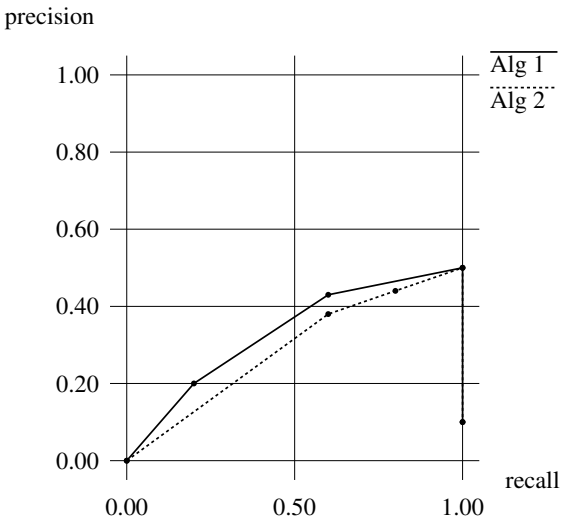


Abbildung 3.7: Recall-Precision-Graphen zu Tab. 3.3

jedem einfachen Recall-Punkt berechnet. Bei n relevanten Dokumenten sind das die Punkte $1/n \ldots n/n$. Für die übrigen Recall-Werte wird dann jeweils die Precision des nächstgrößeren einfachen Recall-Punktes angenommen.
Sei l_f der letzte berücksichtigte Rang. Die *ceiling*-Interpolation ist dann definiert durch:

j : Anzahl irrel. Dok. in Rängen $1 \ldots l_{f-1}$
s : Anzahl rel. Dok., die aus Rang l_f gezogen werden
r : Anzahl rel. Dok., in Rang l_f
i : Anzahl irrel. Dok., in Rang l_f

$$PRECALL_{ceiling}(x) := \frac{\lceil x \cdot n \rceil}{\lceil x \cdot n \rceil + j + (s \cdot i)/r}$$

Beispiel:
Untersuchen wir $\triangle_1 = (+--|+++-------)$ *beim Recall-Level 0.25, d.h. nach dem ersten Rang:*

$$PRECALL_{ceiling}(0.25) = \frac{1}{1 + 0 + (1 \cdot 2)/1} = 1/3$$

Nach der Berechnung von $PRECALL_{ceiling}$ muß wie bei der linearen Ordnung interpoliert werden, indem das Maximum über die bis zu $r = 0$ verlängerten Geradenstücke gebildet wird. Der Leser möge sich überlegen, daß sich dadurch bei Δ_1 nichts ändert, wohl aber bei einem Ergebnis $\triangle_4 = (+--|+++----)$.

3.7 Interpretation von Recall-Precision-Graphen

Die bislang gezeigten R-P-Graphen sind nicht eindeutig zu interpretieren: Sie geben weder die zu erwartende Precision für einen vorgegebenen Recall-Wert eindeutig an, noch sind die durch die Interpolation mittels Geradenstücken entstehenden Zwischenwerte sinnvoll zu interpretieren. Einzig eine probabilistische Interpretation der zu bestimmenden Maße hilft hier weiter.
Bei schwach geordneten Antwortmengen sind zwei Benutzerstandpunkte denkbar. Der Benutzer zieht solange Dokumente aus dem höchsten noch nicht vollständig untersuchten Rang, bis er entweder genug Dokumente (ND) oder ausreichend viele relevante Dokumente (NR) gesehen hat. Wir betrachten im folgenden beide Standpunkte.

3.7.1 Abbruchkriterium: Anzahl der relevanten Dokumente (NR)

Da die Reihenfolge der Dokumente in einem Rang zufällig ist, müssen wir zunächst die erwartete Suchlänge für eine vorgegebene Anzahl relevanter Dokumente bestimmen.

3.7.1.1 Erwartete Suchlänge (esl)

Wir nehmen an, der Benutzer zieht aus einer Menge von r relevanten und i nicht relevanten Dokumenten solange Dokumente, bis er s relevante (mit $s \leq r$) Dokumente hat. esl_s sei die zu erwartende Anzahl von nicht relevanten Dokumenten, die er bis dahin zieht.

$$esl_s := \sum_{v=0}^{i} vP'(v)$$

An Stelle dieses Erwartungswertes berechnen wir zunächst den Erwartungswert $E(X_{r,i,s})$ für die Gesamtzahl der gezogenen Dokumente (wobei $E(X_{r,i,s}) = esl_s + s$).
Dieser Erwartungswert läßt sich als

$$E(X_{r,i,s}) = \sum_{v=s}^{i+s} P(v) \cdot v$$

berechnen. Dabei ist $P(v)$ die Wahrscheinlichkeit, daß man mit den ersten $v-1$ Dokumenten $s-1$ relevante Dokumente zieht und als v-tes Element ein weiteres relevantes. Nachfolgend geben wir die Herleitung aus [Hartmann 86] wieder (siehe auch [Cooper 68]).

$$\begin{aligned}
E(X_{r,i,s}) &:= \sum_{v=s}^{i+s} \frac{\binom{i}{v-s}\binom{r}{s-1}}{\binom{r+i}{v-1}} \frac{r-s+1}{r+i-v+1} v \\
&= \sum_{v=s}^{i+s} \frac{\binom{i}{v-s}\binom{r}{s}}{\binom{r+i}{v}} s \\
&= \sum_{v=s+1}^{i+s+1} \frac{\binom{i}{v-s-1}\binom{r}{s}}{\binom{r+i}{v-1}} s \\
\frac{r+1}{(r+i+1)s} E(X_{r,i,s}) &= \sum_{v=s+1}^{i+s+1} \frac{\binom{i}{v-s-1}\binom{r+1}{s}}{\binom{r+i+1}{v-1}} \frac{r-s+1}{r+i+1-v+1} \\
\frac{r+1}{(r+i+1)s} E(X_{r,i,s}) &= \sum_{v=s+1}^{i+s+1} P(X_{r+1,i,s+1} = v) \\
E(X_{r,i,s}) &= \frac{r+i+1}{r+1} s \\
esl_s &= \frac{i}{r+1} s
\end{aligned}$$

3.7.1.2 Wahrscheinlichkeit (Probability) der Relevanz (PRR)

Eine mögliche Definition von Precision ist die Wahrscheinlichkeit $P(rel|retr)$, daß ein untersuchtes Dokument relevant ist. Für die Anzahl gefundener Dokumente setzen wir jetzt die Summe $NR + esl_{NR}$ (mit $esl_{NR} = j + esl_s$) in die normale Formel zur Berechnung der Precision ein:

$$\begin{aligned} NR &: \text{Anzahl gewünschte relevante Dokumente} \\ PRR &:= \frac{NR}{NR + esl_{NR}} \\ &= \frac{NR}{NR + j + (s \cdot i)/(r+1)} \end{aligned}$$

Indem wir für NR auch reelle Zahlen zulassen, haben wir eine intuitive Methode der Interpolation (siehe die theoretische Rechtfertigung hierzu in [Raghavan et al. 89])

$$\begin{aligned} PRR(x) &:= \frac{x \cdot n}{x \cdot n + esl_{x \cdot n}} \\ &= \frac{x \cdot n}{x \cdot n + j + (s \cdot i)/(r+1)} \end{aligned}$$

Beispiel:
Für unser Retrievalergebnis von oben erhalten wir für den Recall-Level 0.25, also NR=1, folgendes Ergebnis:

$$\triangle = (+--|+++-------)$$
$$PRR = \frac{1}{1 + 0 + (1 \cdot 2)/(1+1)} = 1/2$$

3.7.1.3 Erwartete Precision

Eine andere Möglichkeit, die Definition von Precision auf schwache Rangordnungen zu erweitern, ist die *erwartete Precision*, also der Erwartungswert der Precision. Bezeichne v eine mögliche Anordnung, so berechnet sich die erwartete Precision zu:

$$EP_{NR} := \sum_v P(v)p(v)$$

Beispiel:

$$\triangle = (+--|+++-------) \tag{3.6}$$

$$\begin{aligned} EP_1 &= 1 \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3} + \frac{1}{3} \cdot \frac{1}{3} \\ &= \frac{11}{18} \approx 0.611 \end{aligned}$$

3.7.2 Abbruchkriterium: Anzahl der Dokumente

Ist die Anzahl der Dokumente das Abbruchkriterium, so fallen PRR und EP zusammen:

t_r : Anzahl relevante Dok. in den Rängen $1 \ldots l_{f-1}$
k : Anzahl Dok., die aus Rang l_f gezogen werden

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

Definieren wir analog den *erwarteten Recall* ER

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

so können wir den Graphen der Punkte $\{(ER(ND), EP(ND)\}$ definieren. Dieser stimmt mit dem PRECALL-Graphen mit *intuitiver* Interpolation überein.

$$PRECALL_{intuitive}(x) := \frac{x \cdot n}{x \cdot n + j + (s \cdot i)/r}$$

Die folgenden Abbildungen zeigen die Recall-Precision-Graphen, die sich bei der Anwendung der verschiedenen vorgestellten Methoden ergeben. Die beiden Ergebnisse mit linearer Ordnung sind dabei

$$\begin{aligned} \Delta &= (+|+|-|+|-|+|-|-|-|-|-|-|+|-) \quad \text{und} \\ \Delta &= (+|+|+|-|-|-|-|-|-|-|-|+|-|+|-), \end{aligned}$$

die Ergebnisse zur schwachen Ordnung sind

$$\begin{aligned} \Delta &= (-|++--------|++++|++++--|(80)-) \quad \text{und} \\ \Delta &= (-|++++++-----------|++|++|(80)-). \end{aligned}$$

3.8 Mittelwertbildung und Signifikanztests bei Rangordnungen

Möchte man die in den vorigen Abschnitten beschriebenen Maße nicht nur auf einzelne Fragen anwenden, sondern Mittelwerte bestimmen, so stellt sich im Vergleich zu den ungeordneten Antwortmengen ein neues Problem: Welches ist die geeignete Vergleichsbasis, über der gemittelt wird? Beim booleschen Retrieval wird ja immer die Menge der gefundenen Dokumente betrachtet, aber bei einer Rangordnung existiert keine derartige klare Einteilung. Wenn man sich auf die Makrobewertung beschränkt (Mikrobewertung wird hier fast nie angewandt), so gibt es folgende Möglichkeiten:

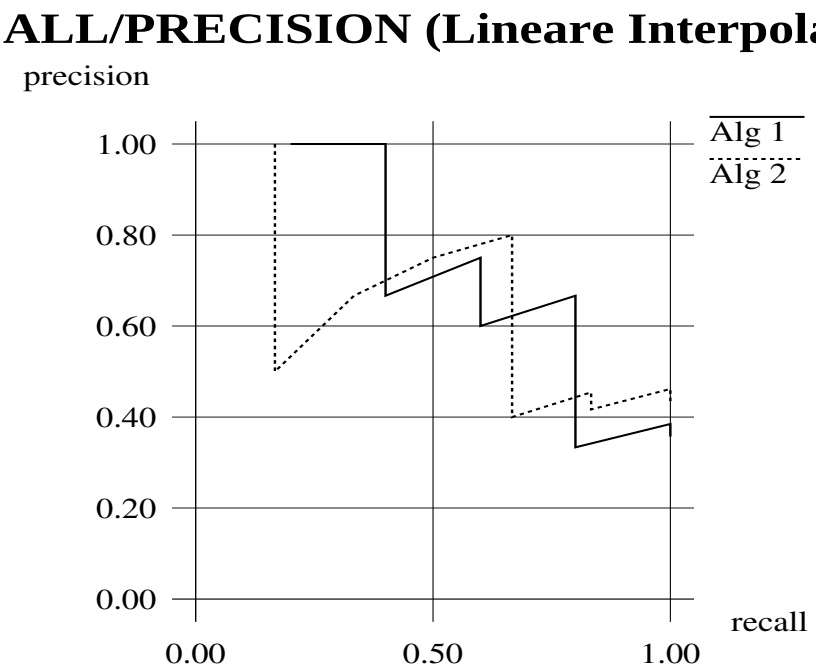


Abbildung 3.8: Lineare Interpolation bei linearer Ordnung

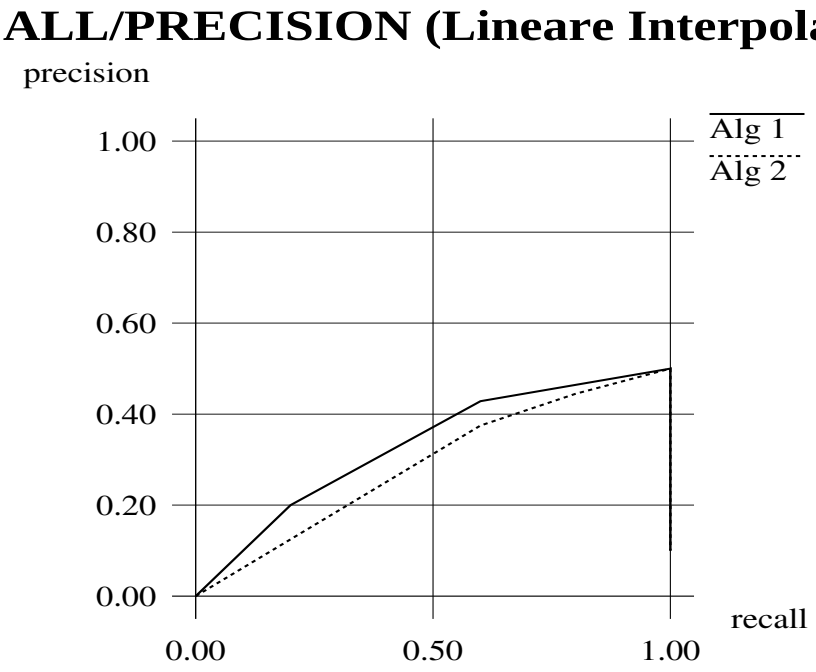


Abbildung 3.9: Lineare Interpolation bei schwacher Ordnung

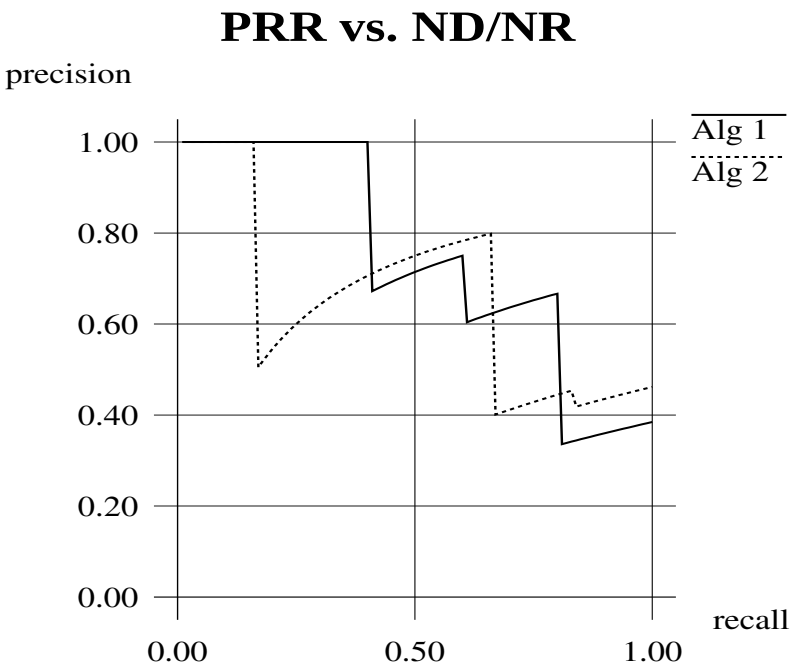


Abbildung 3.10: PR vs. ND bei linearer Ordnung

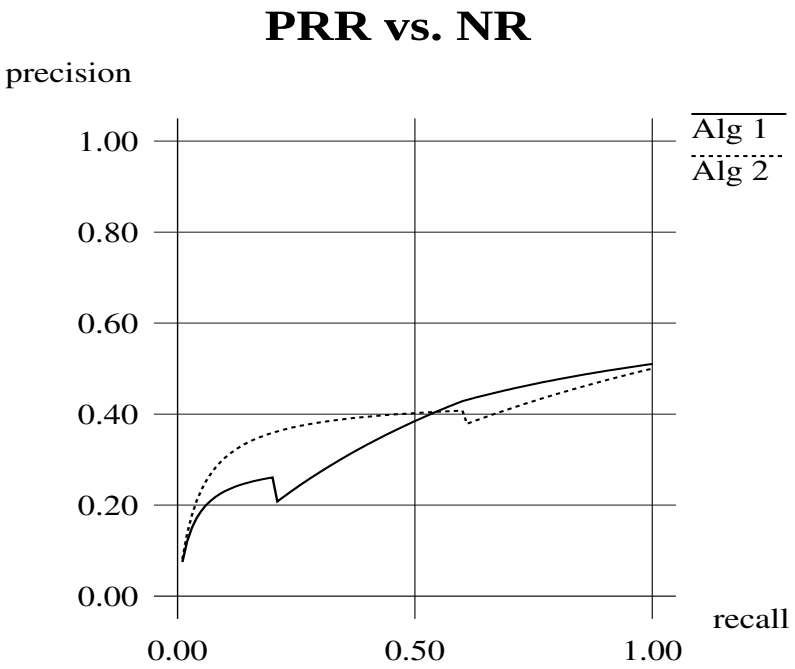


Abbildung 3.11: PR vs. ND bei schwacher Ordnung

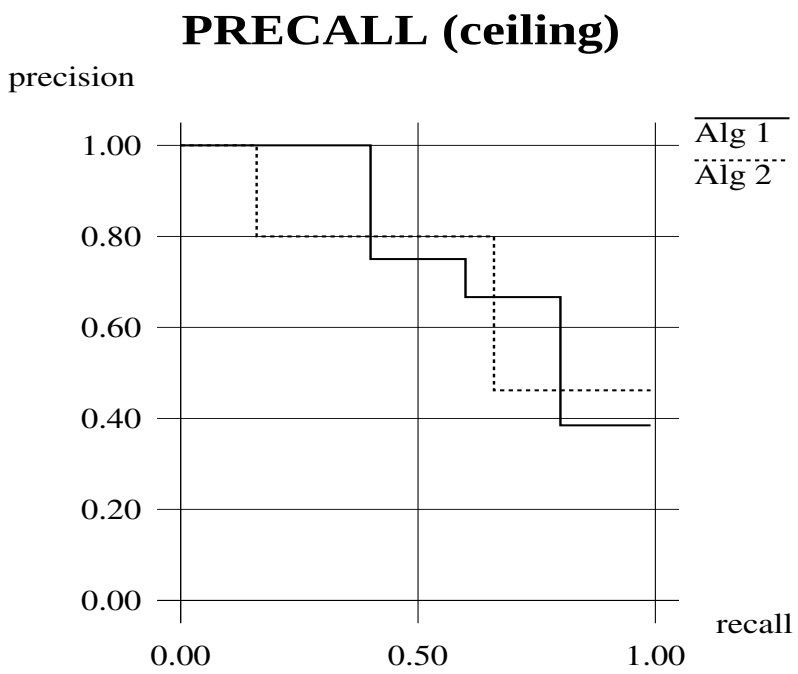


Abbildung 3.12: PRECALL(ceiling) bei linearer Ordnung

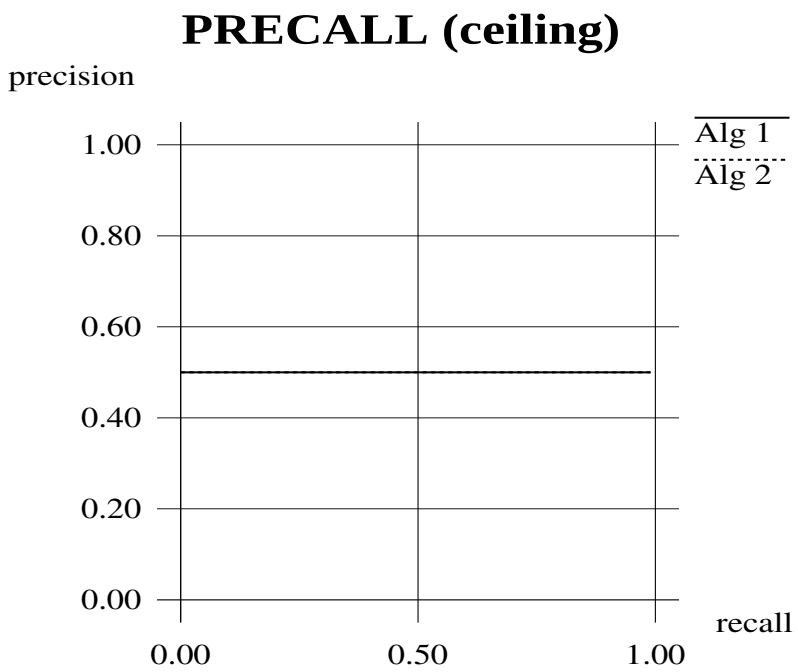


Abbildung 3.13: PRECALL(ceiling) bei schwacher Ordnung

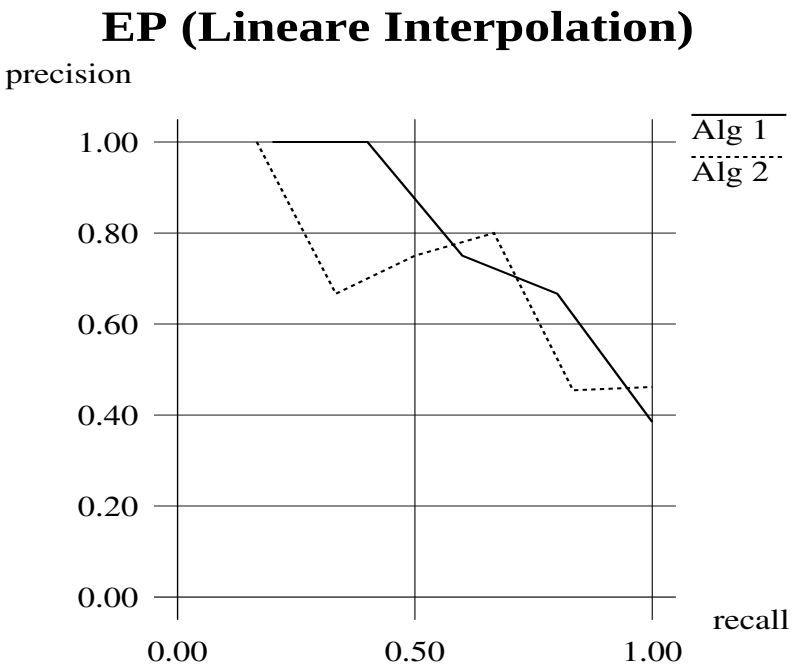


Abbildung 3.14: EP lineare Interpolation bei linearer Ordnung

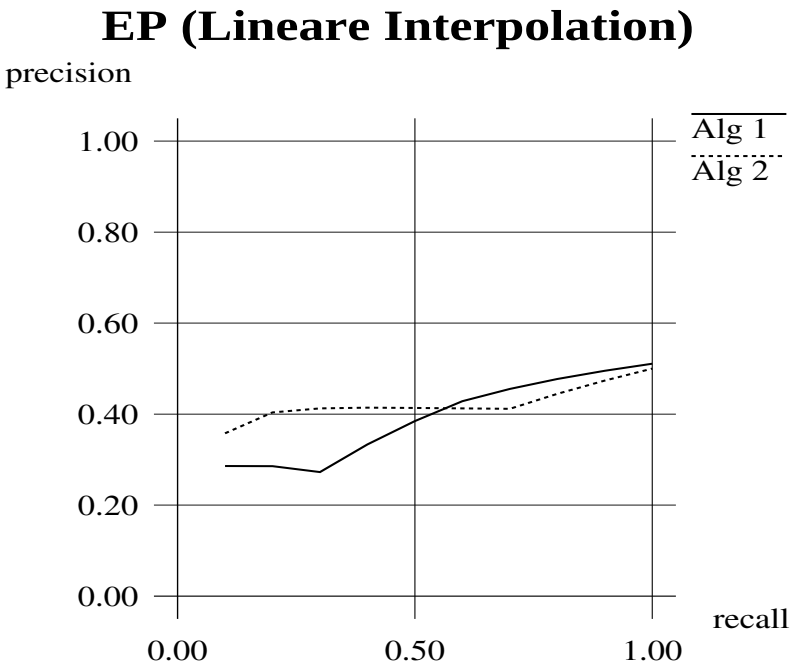


Abbildung 3.15: EP lineare Interpolation bei schwacher Ordnung

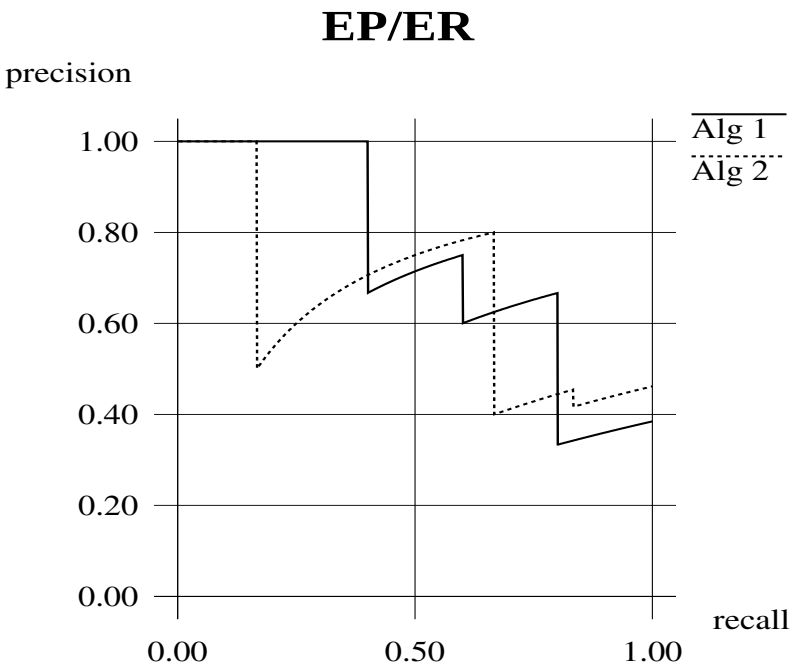


Abbildung 3.16: EP/ER bei linearer Ordnung

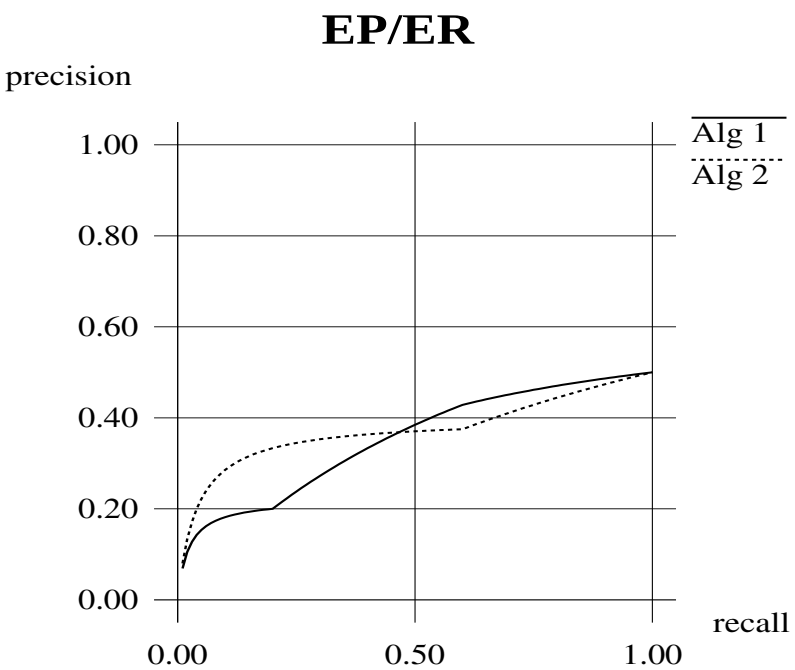


Abbildung 3.17: EP/ER bei schwacher Ordnung

1. Man mittelt über der gleichen Anzahl gefundener Dokumente, nachdem man die Maße nach dem Abbruchkriterium „Anzahl Dokumente (ND)“ berechnet hat. Problematisch bei dieser Vorgehensweise ist, wenn man über Fragen mit unterschiedlicher Generality (relativer Anteil relevanter Dokumente an der Datenbasis) mitteln will.
2. Meist mittelt man über gleichem Recall: Zunächst legt man die Recall-Punkte fest (z.B. in Schritten von 0.1 von 0.1 bis 0.9), für die gemittelt werden soll. Anschließend wird für jede Frage der daraus resultierende Wert NR bestimmt (der nun meist nicht-ganzzahlig ist) und dann entweder PRR oder EP für diesen Wert berechnet, bevor gemittelt wird.

Bei den Evaluierungen mit dem System SMART von Salton [Salton & McGill 83, S. 118-156], die wir hier häufiger zitieren werden, wird meist über die Recall-Werte 0.25, 0.5 und 0.75 gemittelt und dann nochmals das arithmetische Mittel dieser drei Werte gebildet.
Eine alternative Methode zur Betrachtung von Mittelwerten beim Vergleich verschiedener Verfahren ist die Anwendung von Signifikanztests. Auch hierbei muß zunächst eine Vergleichsbasis festgelegt werden (entweder Anzahl gefundene Dokumente oder Recall-Level). Anschließend wird zum Vergleich der sich hierzu ergebenden Precision- (bzw. Recall-) Werte ein Signifikanztest angewendet. Bei verbundenen Stichproben (also gleiche Fragen, die mit verschiedenen Verfahren bearbeitet werden) kann man z.B. den Vorzeichentest oder den Wilcoxon-Test anwenden.

3.9 Nützlichkeitsmaß

Das Nützlichkeitsmaß von Frei und Schäuble [Frei et al. 91] ist eine Alternative zur Bewertung mit Recall und Precision. Dabei wird vor allem die Problematik der Recall-Bestimmung umgangen. Im Gegensatz zur Recall-Precision-Bewertung ist das Nützlichkeitsmaß aber ein rein systemorientiertes Maß, d.h. ein normaler Benutzer des Systems beobachtet nicht direkt die durch dieses Maß ausgedrückte Qualität. Ein weiterer Vorteil des Nützlichkeitsmaßes ist die Eigenschaft, daß die statistische Signifikanz für die gemessene Größe gleich mitberechnet wird, was bei Recall und Precision nicht der Fall ist.
Das Nützlichkeitsmaß dient zum Vergleich zweier Retrievalverfahren (auf der Basis von deren Antwortmengen). Im Unterschied dazu vergleichen die bisher vorgestellten Maße eine Retrievalantwort mit der jeweiligen idealen Antwort. Das Nützlichkeitsmaß erlaubt außerdem, anstelle binärer Relevanzurteile auch Benutzerpräferenzen zur Beurteilung zu verwenden.
Die Größe der Antworten beider Algorithmen (A und B) zu einer Frage sei durch r beschränkt. Sei R die Vereinigung der Dokumente dieser Antworten. Durch die Rangordnung werden auf der Menge der Dokumente in der Datenbank die Ordnungen

π_A und π_B erzeugt. Der Benutzer wird nun aufgefordert, seine Präferenzen (π_p) unter den R Dokumenten der Antwort anzugeben.
Dabei bedeutet ein Paar $(d_1, d_2) \in \pi_p \equiv d_1 <_p d_2$, daß der Benutzer Dokument d_1 für weniger nützlich hält als Dokument d_2
Sei X eine Zufallsvariable, die den Anteil der übereinstimmenden Paare auf $R \times R$ abzüglich des Anteils der invers übereinstimmenden Paare zwischen π_A und π_p angibt. Y bezeichne die entsprechende Zufallsvariable für das Ergebnis B:

$$\begin{aligned} X &:= \frac{|R^2 \cap \pi_p \cap \pi_A| - |R^2 \cap \pi_p^{-1} \cap \pi_A|}{|R^2 \cap \pi_p|} \\ Y &:= \frac{|R^2 \cap \pi_p \cap \pi_B| - |R^2 \cap \pi_p^{-1} \cap \pi_B|}{|R^2 \cap \pi_p|} \end{aligned}$$

Berechnet man die Werte dieser Zufallsvariablen für mehrere (k) Anfragen, so erhält man k Paare $\{(x_i, y_i) | 0 \leq i < k\}$, auf die man folgenden Algorithmus anwendet:

1. Berechne die Differenzen $y_i - x_i$ und entferne die Paare, deren Differenz 0 ist (k_0 sei die Anzahl der verbleibenden Paare).
2. Sortiere die Differenzen nach aufsteigenden Beträgen. (Haben mehrere Differenzen den gleichen Betrag, weise ihnen den mittleren Rang zu)
3. Berechne w_+, die Summe der Ränge mit positiven Vorzeichen

Der Erwartungswert μ für w_+ ist gegeben durch:

$$\mu = \frac{k_0(k_0+1)}{4}$$

Das Nützlichkeitsmaß $u_{A,B}$ sei definiert durch

$$u_{A,B} := \frac{w_+ - \mu}{\mu}$$

Um die Paare (x_i, y_i) mit Differenz 0 und die Größe der Differenz zu berücksichtigen, definieren wir als korrigiertes Nützlichkeitsmaß:

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

Große Werte von $u_{A,B}$ weisen darauf hin, daß Algorithmus B besser ist als Algorithmus A. Um die statistische „Sicherheit“ dieses Hinweises festzustellen, benötigen wir die Wahrscheinlichkeit $P_k(U_{A,B} \geq u_{A,B})$, daß ein so großer Wert von $U_{A,B}$ zufällig ist.

Für die Berechnung nehmen wir an, daß $U_{A,B}$ normal verteilt ist:

$$\begin{aligned} P_k(U_{A,B} \geq u_{A,B}) &\approx 1 - \Phi\left(\frac{w_+ - \mu}{\sigma}\right) \\ \mu &= \frac{k_0(k_0+1)}{4} \\ \sigma^2 &= \frac{k_0(k_0+1)(2k_0+1)}{24} \end{aligned}$$

Beispiel
Die folgenden Tabellen zeigen die Berechnung des Nützlichkeitsmaßes für vier Fragen.

	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		

Die Antwortmengen:

query	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

query	$R^2(D, q, r) \cap \pi_p$
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:

query	$R^2(D,q,r) \cap \pi_p \cap \pi_A$	$R^2(D,q,r) \cap \pi_p \cap \pi_B$
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:

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

Die Werte x_i, y_i und $y_i - x_i$:

query	x_i	y_i	$y_i - x_i$
q_0	0	1	1
q_1	1	1	0
q_2	1	-1	-2
q_3	-1	1	2

Die Ränge der Differenzbeträge $|y_i - x_i|$

i	$\|y_i - x_i\|$	sign	rank
0	1	+1	1
2	2	–1	2.5
3	2	+1	2.5

Damit ergeben sich folgende Werte:

$$\begin{aligned} w_+ &= 7/2 \\ \mu &= 3 \\ \sigma^2 &= 7/2 \\ u_{A,B} &= 1/6 \\ u^*_{A,B} &= \tfrac{1}{24} \end{aligned}$$

$$P_k(U_{A,B} \geq u_{A,B}) \approx 1 - \Phi(1/\sqrt{14}) = 0.4$$

Kapitel 4

Wissensrepräsentation für Texte

4.1 Problemstellung

Da sich IR hauptsächlich mit der inhaltlichen Suche in Texten beschäftigt, stellt sich die Frage nach der geeigneten Repräsentationsform für Textinhalte. Im Gegensatz zu Standard-Datenbanksystemen, wo die Repräsentation mehr oder weniger eindeutig ist, ist die Repräsentation ein zentrales Problem im IR. Wenn wir an unsere Beispiel-Anfrage aus der Einführung denken (*„side-effects of drugs on memory“*), so wird klar, daß die in einer Frage angesprochenen Konzepte auf unterschiedlichste Weise in Texten formuliert sein können. Eine gewählte Repräsentationsform soll daher zum einen unterschiedliche Formulierungen auf die gleiche Repräsentation abbilden (und damit den Recall erhöhen), zum anderen auch unklare Formulierungen (z.B. Mehrdeutigkeit einzelner Wörter) vereindeutigen, um die Precision zu erhöhen.
Wir werden im folgenden verschiedene Lösungsansätze für dieses Problem vorstellen:

- Freitextsuche:
  Hierbei wird keine zusätzliche Repräsentation erstellt, sondern es werden nur bestimmte Funktionen zur Verbesserung der Suche im Text der Dokumente angeboten. Zu unterscheiden ist hierbei zwischen zwei Ansätzen:
  - informatischer Ansatz:
    Dieser Ansatz (der in den heute kommerziell angebotenen IR-Systemen fast ausschließlich vertreten ist) faßt Textretrieval als Zeichenkettensuche auf und bietet entsprechende Funktionen auf Zeichenkettenebene.
  - computerlinguistischer Ansatz:
    Hier wird mit Hilfe von morphologischen und teilweise auch syntaktischen Verfahren eine Normalisierung von Wortformen angestrebt, so daß sich die Suche auf Wörter bezieht (im Gegensatz zu den Zeichenketten beim informatischen Ansatz).
- semantischer Ansatz:
  Durch die Zuordnung von Deskriptionen zu Texten wird versucht, eine Re-

präsentation zu erstellen, die weitgehend unabhängig von der konkreten Formulierung im Text ist. Syntax und Semantik solcher Deskriptionen sind in Form sogenannter Dokumentationssprachen festgelegt.

4.2 Freitextsuche

Bei der Freitextsuche werden zunächst folgende Verarbeitungsschritte auf den Text der Dokumente angewandt:

1. Zerlegung des Textes in einzelne Wörter: Leer- und Interpunktionszeichen werden hier als Worttrenner aufgefaßt.
2. Stoppworteliminierung: Nicht-bedeutungstragende Wörter wie Artikel, Füllwörter oder Konjunktionen werden meist aus Aufwandsgründen von der weiteren Verarbeitung ausgeschlossenn. Stoppwörter machen häufig rund die Hälfte des Textes aus.
3. Satzendeerkennung: Einige Freitextfunktionen erlauben den Bezug auf Satzgrenzen, die folglich erst erkannt werden müssen. Wegen der Verwechslungsmöglichkeit mit Abkürzungspunkten kann diese Aufgabe nur approximativ gelöst werden (z.B. mit Hilfe von Abkürzungslisten).

Die eigentliche Freitextsuche bezieht sich dann auf den so reduzierten Text (bzw. die resultierende Folge von Wörtern). Bei dieser Art der Suche nach Wörtern stellen sich folgende Probleme:

- **Homographen** (verschieden gesprochene Wörter mit gleicher Schreibweise)
  Tenor: Sänger / Ausdrucksweise
- **Polyseme** (Wörter mit mehreren Bedeutungen)
  Bank: Sitzgelegenheit / Geldinstitut
- **Flexionsformen**, die durch Konjugation und Deklination eines Wortes entstehen
  Haus – (des) Hauses – Häuser,
  schreiben – schreibt – schrieb – geschrieben
- **Derivationsformen** (verschiedene Wortformen zu einem Wortstamm)
  Formatierung – Format – formatieren
- **Komposita** (mehrgliedrige Ausdrücke)
  Bundeskanzlerwahl – Wahl des Bundeskanzlers
  information retrieval – retrieval of information – information was retrieved

Das grundsätzliche Problem der Freitextsuche — die Wortwahl — bleibt aber in jedem Falle ungelöst!

4.2.1 Informatischer Ansatz

Der informatische Ansatz betrachtet Texte als Folgen von Wörtern, wobei ein Wort als eine durch Leer- oder Interpunktionszeichen begrenzte Zeichenfolge definiert ist. Somit wird hier Freitextsuche als eine spezielle Form der Zeichenkettensuche aufgefaßt und entsprechende Zeichenketten-Operatoren angeboten. Diese beziehen sich zum einen auf einzelne Wörter, zum anderen auf Folgen von Wörtern. Erstere sind Truncation- und Maskierungs-Operatoren für die Freitextsuche, letztere die Kontextoperatoren. (Wie bei allen IR-Systemen üblich, wird im folgenden nicht zwischen Groß- und Kleinschreibung unterschieden).

- Truncation- und Maskierungs-Operatoren dienen dazu, Flexions- und Derivationsformen von Wörtern zusammenzuführen.
  - Bei der **Truncation** wird einerseits zwischen Front- und End-Truncation unterschieden, wobei die Front-Truncation hauptsächlich benutzt wird, um beliebige Vorsilben bei der Suche zuzulassen. Andererseits kann bei der Truncation entweder eine feste oder eine variable Anzahl von Zeichen zugelassen werden. Bei den folgenden Beispielen verwenden wir das Symbol $ für Truncation für genau ein Zeichen und # für eine beliebig lange Zeichenfolge; im ersten Fall spricht man auch von beschränkter Truncation, im zweiten Fall von unbeschränkter. Wir geben jeweils das Suchpattern an und einige Wörter, die Treffer für dieses Pattern sind:
    schreib#: schreiben, schreibt, schreibst, schreibe
    schreib$$: schreiben, schreibst
    #schreiben: schreiben, beschreiben, anschreiben, verschreiben
    $$schreiben: beschreiben, anschreiben
  - **Maskierung** oder genauer Mitten-Maskierung bezieht sich auf Zeichen in der Mitte eines Wortes; da im Deutschen bei der Konjugation und der Deklination von Wörtern nicht nur die Endung betroffen ist, werden solche Operationen benötigt:
    schr$$b#: schreiben, schrieb / schrauben
    h$$s#: Haus, Häuser / Hanse, hausen, hassen

Der wesentliche Vorteil der Truncation- und Maskierungsoperatoren besteht also darin, daß Flexions- und Derivationsformen von Wörtern zusammengeführt werden und Schreibarbeit gegenüber dem expliziten Aufzählen gespart wird. Möglicherweise werden dadurch aber auch unerwünschte Wörter zugelassen; daher zeigen die meisten Systeme zunächst die verschiedenen Wortformen, die ein Pattern erfüllen, so daß der Benutzer daraus auswählen kann. Das grundsätzliche Problem bei dieser Vorgehensweise ist aber, daß der Benutzer sich zunächst alle möglichen Wortformen vorstellen muß, um eine gute Anfrage zu formulieren.

- **Kontextoperatoren** dienen zur Suche nach mehrgliedrigen Ausdrücken. Da z.B. der Ausdruck “information retrieval” im Text auch in der Form “infor-

mation storage and retrieval" oder "retrieval of information" auftreten kann, muß die Anfragesprache Operatoren anbieten, die die einfache Spezifikation solcher Formen ermöglichen. Ohne solche speziellen Operatoren wäre man auf die booleschen Operatoren angewiesen, die sich lediglich auf das Vorkommen der einzelnen Wörter irgendwo im selben Text beziehen. Folgende Kontextoperatoren werden häufig angeboten:

- genauer Wortabstand ($):
  retrieval $ information: retrieval of information, retrieval with information loss
- maximaler Wortabstand (#):
  text # # retrieval: text retrieval, text and fact retrieval
- Wortreihenfolge (,):
  information # , retrieval: information retrieval, retrieval of information
- gleicher Satz (.):
  information # retrieval. matcht nicht
  ... this information. Retrieval of data ...
  aber auch nicht:
  ... storage of information. Its retrieval ...

4.2.2 Computerlinguistischer Ansatz

Der computerlinguistischer Ansatz versucht, Verfahren bereitzustellen, die die verschiedenen Flexions- und Derivationsformen eines Wortes zusammenführen. Analog sollen bei mehrgliedrigen Ausdrücken die verschiedenen möglichen Vorkommensformen erkannt werden. Im Gegensatz zum informatischen Ansatz, der zur Bewältigung dieser Probleme nur recht primitive Hilfsmittel zur Verfügung stellt, werden beim computerlinguistischen Ansatz Algorithmen bereitgestellt, die diese Transformationen automatisch ausführen. Dabei ist allerdings zu beachten, daß diese Aufgabe nicht in perfekter Art und Weise gelöst werden kann.
Es gibt folgende Arten von computerlinguistischen Verfahren:

- **graphematische Verfahren** basieren auf der Analyse von Buchstabenfolgen und werden im Bereich der Morphologie zur Zusammenführung von Flexions- oder Derivationsformen eines Wortes eingesetzt,
- **lexikalische Verfahren** basieren auf einem Wörterbuch, das zum einen mehrgliedrige Ausdrücke enthalten kann, andererseits die verschiedenen Bedeutungen mehrdeutiger Wörter verzeichnet,
- **syntaktische Verfahren** dienen hauptsächlich zur Identifikation von mehrgliedrigen Ausdrücken.

4.2.2.1 Graphematische Verfahren

In diesem Abschnitt sollen graphematische Algorithmen für die englische Sprache vorgestellt werden. Da das Englische im Gegensatz zum Deutschen nicht so stark flektiert ist, erreichen diese Algorithmen eine sehr hohe Genauigkeit und sind daher ohne Probleme praktisch einsetzbar. Es ist zwischen Grundform- und Stammformreduktion zu unterscheiden:

- Bei der **Grundformreduktion** werden Wörter auf ihre Grundform zurückgeführt. Die Grundform ist bei Substantiven der Nominativ Singular und bei Verben deren Infinitiv. Je nach Art des Algorithmus wird unterschieden zwischen:
  - **formaler Grundform**, die durch das alleinige Abtrennen der Flexionsendung erzeugt wird, wie z.B.
    activities → *activit*
  - und **lexikographischer Grundform**, die durch
    Abtrennen der Flexionsendung und ggfs. anschließender Rekodierung entsteht, also z.B.
    applies → *appl* → *apply*
- Bei der **Stammformreduktion** werden (nach vorheriger Grundformreduktion) die Wörter auf ihren Wortstamm reduziert, indem die Derivationsendungen entfernt werden, z.B.:
  computer, compute, computation, computerization → *comput*

4.2.2.1.1 Lexikographische Grundformreduktion

Als Beispiel für einen Reduktionsalgorithmus soll hier eine vereinfachte Fassung der in [Kuhlen 77] beschriebenen lexikographischen Grundformreduktion vorgestellt werden. Hierzu verwenden wir folgende Notationen:

%	alle Vokale (einschließlich Y)
*	alle Konsonanten
J	Länge des Wortes
/	‚oder'
Ø	Leerzeichen
→	'zu'
←	'aus'
¬	'nicht'

Die Regeln dieses (vereinfachten) Algorithmus' sind dann folgende:

1) **IES** → **Y**
2) **ES** → **Ø** wenn *O / CH / SH / SS / ZZ / X vorangehen
3) **S** → **Ø** wenn * / E / %Y / %O / OA / EA vorangehen

4) **S'** → **Ø**
   IES' → **Y**
   ES' → **Ø**
5) **'S** → **Ø**
   ' → **Ø**
6) **ING** → **Ø** wenn ** / % / X vorausgehen
   ING → **E** wenn %* vorausgehen
7) **IED** → **Y**
8) **ED** → **Ø** wenn ** / % / X vorausgehen
   ED → **E** wenn %* vorausgehen

Der Algorithmus wendet jeweils nur die erste passende Regel an.
Nachfolgend geben wir einige Beispiele zu den einzelnen Regeln.

Regel 1 **IES** → **Y**

Beispiele zu 1:

APPLIES	→	APPLY
IDENTIFIES	→	IDENTIFY
ACTIVITIES	→	ACTIVITY

Regel 2 **ES** → **Ø**, wenn *O / CH / SH / SS / ZZ / X vorangehen

---

Beispiele zu 2:

BREACHES	→	BREACH
PROCESSES	→	PROCESS
FISHES	→	FISH
COMPLEXES	→	COMPLEX
TANGOES	→	TANGO
BUZZES	→	BUZZ

Regel 3 **S** → **Ø**, wenn * / E / %Y / %O / OA / EA vorangehen

---

Beispiele zu 3:

METHODS	→	METHOD
HOUSES	→	HOUSE
BOYS	→	BOY
RADIOS	→	RADIO
COCOAS	→	COCOA
FLEAS	→	FLEA

Regel 4	**S'**	→	**Ø**
	IES'	→	**Y**
	ES'	→	**Ø**

Beispiele zu 4:

MOTHERS'	→	MOTHER
LADIES'	→	LADY
FLAMINGOES	→	FLAMINGO

Regel 5	**'S**	→	**Ø**
	'	→	**Ø**

Beispiele zu 5:

MOTHER'S	→	MOTHER
CHILDREN'S	→	CHILDREN
PETRUS'	→	PETRUS

Regel 6	**ING**	→	**Ø**, wenn ** / % / X vorausgehen
	ING	→	**E**, wenn %* vorausgehen

Beispiele zu 6:

DISGUSTING	→	DISGUST
GOING	→	GO
MIXING	→	MIX
LOOSING	→	LOOSE
RETRIEVING	→	RETRIEVE

Regel 7	**IED**	→	**Y**

Beispiel zu 7:

SATISFIED	→	SATISFY

Regel 8	**ED**	→	**Ø**, wenn ** / % / X vorausgehen
	ED	→	**E**, wenn %* vorausgehen

Beispiel zu 8:

DISGUSTED	→	DISGUST
OBEYED	→	OBEY
MIXED	→	MIX
BELIEVED	→	BELIEVE

4.2.2.2 Lexikalische Verfahren

Graphematische Verfahren sind für stark flektierte Sprachen wie z.B. das Deutsche wenig geeignet. Daher muß man hier lexikalische Verfahren einsetzen. Für den Einsatz im IR sollte ein Lexikon folgende Relationen enthalten (s.a. [Zimmermann 91]):

- Flexionsform (Vollformen) — zugehörige Grundform
  Hauses - Haus, ging - gehen
- Derivationsform — zugehörige Grundformen
  Lieblosigkeit — lieblos, Berechnung — rechnen
- Komposita — zugehörige Dekomposition
  Haustür — Tür, Armbanduhr — Uhr.

Lexikalische Verfahren haben allerdings den Nachteil, daß hier eine ständige Pflege des Wörterbuches notwendig ist. Für eine neue Anwendung ist zunächst ein hoher Anpassungsaufwand notwendig, um ein Standard-Wörterbuch mit den jeweiligen Fachbegriffen anzureichern. Auch später tauchen ständig neue Begriffe auf, die in das Lexikon aufgenommen werden müssen.

4.2.2.3 Syntaktische Verfahren

Syntaktische Verfahren werden im IR hauptsächlich zur Identifikation von mehrgliedrigen Ausdrücken (Komposita) eingesetzt. Entsprechend der Aufgabenstellung sind hierfür bestimmte Arten von Parsern besonders geeignet:

- **Wörterbuchunabhängige Parser** arbeiten ohne (umfangreiches) Wörterbuch. Die meisten Parser basieren auf der Annahme, daß ein Wörterbuch vorhanden ist, in dem alle im Text auftretenden Wörter verzeichnet sind. Ein solcher Ansatz würde allerdings für praktische Anwendungen einen enorm hohen Pflegeaufwand erfordern, der nur in Ausnahmefällen zu leisten ist.
- **Robuste Parser** ignorieren auch Fehler in der Eingabe, wie z.B. Schreibfehler oder ungrammatische Sätze; insbesondere können auch sprachlich korrekte Sätze auftreten, die aber vom Parser nicht verstanden werden (weil z.B. die Grammatik nicht vollständig ist). Ein robuster Parser sollte beim Auftreten solcher Sätze zum einen nicht „abstürzen“, zum anderen noch eine möglichst sinnvolle Ausgabe liefern.

4.2.2.3.1 Der PLNLP-Parser

Der PLNLP-Parser [Heidorn et al. 82] ist ein robuster, wörterbuchunabhängiger Parser für die englische Sprache. In [Salton & Smith 89] wird die Anwendung dieses Systems zur Identifikation von Nominalphrasen diskutiert. Wir wollen hier nur zwei Beispiele für die Ausgabe dieses Parsers zeigen, wobei in Abb. 4.1 ein Satz korrekt analysiert wird, in Abb. 4.2 dagegen fehlerhaft.

DECL	PP	PREP	"by"					
		VERB*	"eliminating"					
		NP	NOUN*	"redundancies"				
			PUNC	"--"				
			NAPPOS	DET	ADJ*	"a"		
				NOUN*	"method"			
				PTPRTCL	VERB*	"known"		
					PP	PREP	"as"	
						NP	NOUN*	"text"
						NOUN*	"compression"	
						PUNC	","	
	NP	PRON*	"it"					
	VERB*	"is"						
	AJP	AVP	ADV*	"often"				
	ADJ*	"possible"						
		INFCL	INFTO	"to"				
			VERB*	"reduce"				
			NP	NP	NOUN*"	"text"		
				NOUN*	"sizes"			
				PP	AVP	ADV*	"considerably"	
					PREP	"without"		
					QUANT	ADJ*	"any"	
					NOUN*	"loss"		
					PP	PREP	"of"	
						NP	NOUN*	"text"
						NOUN*	"content"	

Abbildung 4.1: Beispiel für korrektes Parsing

CMPD	DECL	NP	NP	NOUN*	"today"		
			AJP	ADJ*	"large"		
			NP	NOUN*	"disk"		
			NOUN*	"array"			
		VERB*	"are"				
		AJP	AVP	ADV*	"usually"		
			ADJ*	"available			
	CONJ*	", but"					
	DECL	NP	AJP	ADJ*	"using"		
			AJP	ADJ*	"short"		
			NP	NP	NOUN*	"texts"	
				CONJ*	"and"		
				NP	NOUN*	"small"	
			NOUN	"dictionary"			
		VP	VERB*	"sizes"			
			NP	NOUN*	"saves"		
				PRPRTCL	VERB*	"processing"	
					NP	NOUN*	"time"
						PP	PREP
							NOUN*
							PP
		CONJ*	"and"				
		VP	AVP	ADV*	"still"		
			VERB*	"remains"			
			AJP	ADJ*	"attractive"		

Abbildung 4.2: Beispiel für eine fehlerhafte syntaktische Analyse

4.2.2.3.2 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 erhoben:

1. Abstand zwischen den Komponenten (wobei bei der Bestimmung des Wortabstandes entweder 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 Wahrscheinlich-

keit 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 kann man sagen, daß die vorletzte Mehrwortgruppe "MEASURE ELECTRICAL" zwar syntaktisch falsch ist, aber semantisch korrekt, so daß bezogen auf das Retrieval dies kein eigentlicher Fehler ist.) 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[1]. 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.

4.3 Dokumentationssprachen

4.3.1 Allgemeine Eigenschaften

Dokumentationssprachen sollen die im vorangegangenen Abschnitt dargestellten Nachteile der Freitextsuche überwinden helfen. Um sich von der konkreten sprachlichen Formulierung in dem zu indexierenden Dokument zu lösen, wird eine davon unabhängige Repräsentation des Textinhaltes durch Verwendung eines speziellen Vokabulars verwendet. Dieses Vokabular soll alle Mehrdeutigkeiten und die Probleme morphologischer und syntaktischer Art der natürlichen Sprache vermeiden. In den folgenden Abschnitten betrachten wir zunächst zwei „klassische" Arten von Dokumentationssprachen, nämlich Klassifikationen und Thesauri. Diese Ausführungen orientieren sich im wesentlichen an der Darstellung in [Burkart 90]. Anschließend wird als Beispiel für einen KI-orientierten Ansatz die Sprache KL-ONE vorgestellt.

4.3.2 Klassifikationen

Klassifikationen dienen als Strukturierung eines Wissensgebietes nach einem vorgegebenen formalen Schema. Einem einzelnen Dokument wird dabei in der Regel nur eine Klasse zugeordnet. Aus dieser Randbedingung ergibt sich bereits eine prinzipielle Schwäche, da viele Dokumente ja gerade versuchen, Brücken zwischen verschiedenen

[1] Im Darmstädter Indexierungsansatz wird tatsächlich eine etwas andere Vorgehensweise beschritten: Die Gewichte werden nicht explizit bestimmt; stattdessen geht die Vorkommensklasse als Merkmal in ein Regressionsverfahren ein, wo dieses Merkmal erst in Kombination mit anderen Merkmalen gewichtet wird

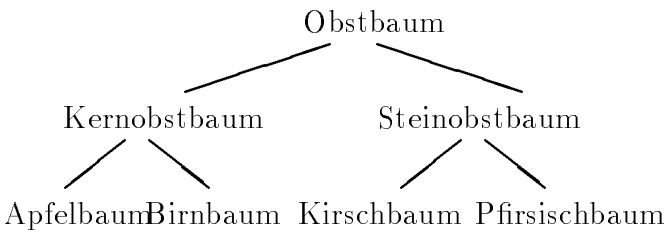


Abbildung 4.3: Monohierarchie

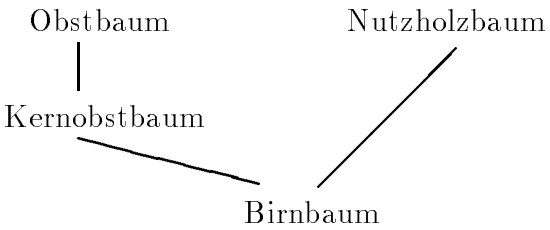


Abbildung 4.4: Polyhierarchie

Wissensgebieten zu schlagen, so daß sie zu mehreren Klassen gehören. Andererseits gibt es einige praktische Anwendungen, die gerade eine eindeutige Klassifikation von Dokumenten voraussetzen, z.B. bei der fachsystematischen Aufstellung von Büchern in einer Bibliothek oder bei der Anordnung von Abstracts in der gedruckten Fassung eines Referateorgans.

4.3.2.1 Eigenschaften von Klassifikationssystemen

Wir betrachten zunächst einige grundlegende Eigenschaften von Klassifikationssystemen, bevor wir konkrete Beispiele vorstellen.

4.3.2.1.1 Monohierarchie — Polyhierarchie

Abbildung 4.3 zeigt eine monohierarchische Klassifikation; hierbei sind die Klassen in eine Baumstruktur eingeordnet. Häufig reicht aber eine Baumstruktur nicht aus, um die Beziehungen zwischen den Klassen sinnvoll darzustellen. Deswegen geht man zu einer Polyhierarchie über, bei der eine Klasse mehrere Superklassen haben kann (Abbildung 4.4).

4.3.2.1.2 Monodimensionalität — Polydimensionalität

Bei der Festlegung der Klassenstruktur kann es häufig auf einer Stufe mehrere Merkmale geben, nach denen eine weitere Aufteilung in Unterklassen vorgenommen werden kann, wobei diese Merkmale orthogonal zueinander sind. Eine polydimensionale

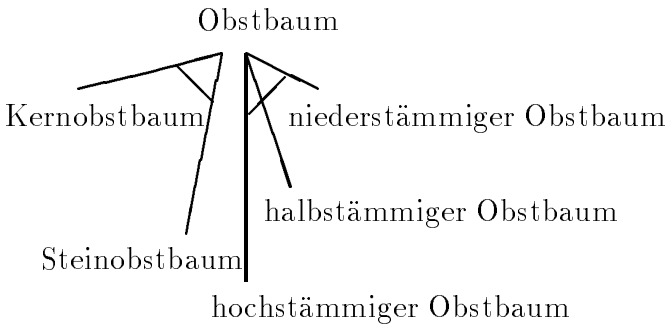


Abbildung 4.5: Polydimensionalität

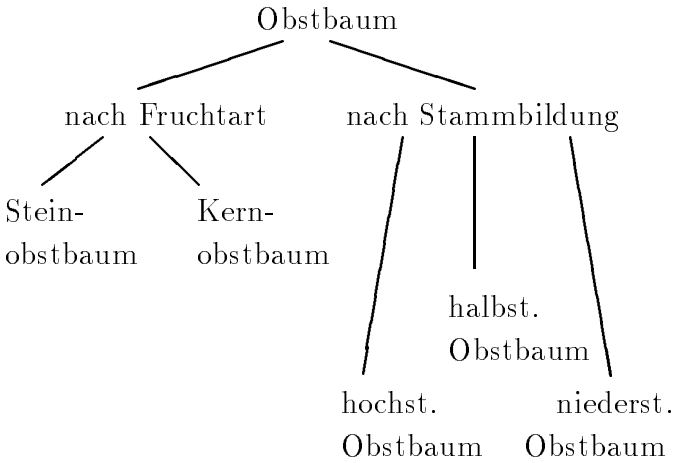


Abbildung 4.6: Aufgelöste Polydimensionalität

Klassifikation, wie am Beispiel in Abb. 4.5 illustriert, erlaubt die Darstellung dieses Sachverhaltes. Erlaubt das Klassifikationsschema keine Polydimensionalität, dann muß diese durch Einführung einer zusätzlichen Hierarchieebene (s. Abb. 4.6) aufgelöst werden, wodurch das Schema unübersichtlicher wird.

4.3.2.1.3 Analytische vs. synthetische Klassifikation

Beim Entwurf eines Klassifikationsschemas gibt es — ähnlich wie bei der Programmierung — zwei mögliche Vorgehensweisen. Die bisherigen Beispiele illustrieren die analytische Klassifikation, die top-down vorgeht: Ausgehend von der Grundgesamtheit der zu klassifizierenden Objekte sucht man rekursiv jeweils nach dem nächsten Kriterium zur weiteren Aufteilung der Objektmenge.

Facette	Facette	Facette
A Fruchtart	B Stammart	C Erntezeit
A1 Apfel	B1 hochstämmig	C1 früh
A2 Birne	B2 halbstämmig	C2 mittel
A3 Kirsche	B3 niederstämmig	C3 spät
A4 Pfirsich		
A5 Pflaume		

Tabelle 4.1: Beispiel zur Facettenklassifikation

Im Gegensatz dazu geht die synthetische Klassifikation bottom-up vor. Dabei werden zuerst die relevanten Merkmale der zu klassifizierenden Objekte erhoben und im Klassifikationssystem zusammengestellt. Im zweiten Schritt werden dann die Klassen durch Kombination der Merkmale gebildet. Die synthetische Klassifikation bezeichnet man auch als **Facettenklassifikation**. Tabelle 4.1 zeigt eine solche Klassifikation für Obstbäume. In diesem Schema würde ein niederstämmiger Frühapfelbaum mit A1B3C1 klassifiziert. Für die Definition der Facetten gelten folgende Regeln:

1. Die Facetten müssen disjunkt sein.
2. Innerhalb einer Facette muß monodimensional unterteilt werden.

Zusätzlich müssen noch syntaktische Regeln definiert werden, die die Bildung der Klassen aus den Facetten festlegen.

4.3.2.2 Dezimalklassifikation

Als bekanntestes Beispiel für Klassifikationssysteme gilt sicher die Dezimalklassifikation. Sie geht auf die Dewey Decimal Classification (DDC) zurück, die 1876 von Melvil Dewey in den USA als Universalklassifikation zur Aufstellung von Buchbeständen konzipiert wurde. Daraus entwickelten dann die Belgier Paul Otlet und Henri Lafontaine

durch das Hinzufügen von syntaktischen Elementen die **Universelle Dezimalklassifikation** (DK), die zur Inhaltserschließung geeignet ist.

4.3.2.2.1 Grundelemente der DK

Wir stellen im folgenden die wesentlichen Grundelemente der DK (Dezimalklassifikation) vor:

- Die Klassen der DK sind hierarchisch gegliedert. Wie der Name schon sagt, ist der maximale Verzweigungsgrad 10. Das gesamte System enthält derzeit über 130000 Klassen.
- Zusätzlich zu diesen Klassen erlauben **Anhängezahlen** die Facettierung.
- Zur Verknüpfung mehrerer DK-Zahlen dienen bestimmte Sonderzeichen.

4.3.2.2.2 Klassen der DK

Die DK-Haupttafeln umfassen folgende 10 Hauptabteilungen:

0 Allgemeines
1 Philosophie
2 Religion, Theologie
3 Sozialwissenschaften, Recht, Verwaltung
4 (zur Zeit nicht belegt)
5 Mathematik, Naturwissenschaften
6 Angewandte Wissenschaften, Medizin, Technik
7 Kunst, Kunstgewerbe, Photographie, Musik, Spiel, Sport
8 Sprachwissenschaft, Philologie, Schöne Literatur, Literaturwissenschaft
9 Heimatkunde, Geographie, Biographien, Geschichte

Diese Hauptklasse werden bis hin zu sehr speziellen Sachverhalten weiter untergliedert, wie etwa im folgenden Beispiel:

3 Sozialwissenschaften, Recht, Verwaltung
33 Volkswirtschaft
336 Finanzen. Bank- und Geldwesen
336.7 Geldwesen. Bankwesen. Börsenwesen
336.76 Börsenwesen. Geldmarkt. Kapitalmarkt
336.763 Wertpapiere. Effekten
336.763.3 Obligationen. Schuldverschreibungen
336.763.31 Allgemeines
336.763.311 Verzinsliche Schuldbriefe
336.763.311.1 Langfristig verzinsliche Schuldbriefe

4.3.2.2.3 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

4.3.2.2.4 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.

4.3.2.3 Computing Reviews Classification

Als zweites Beispiel eines Klassifikationsschemas zeigen wir hier aus dem Bereich der Informatik die Computing Reviews (CR) Classification, die zur Anordnung der Artikel in der Zeitschrift *ACM Computing Reviews* entworfen wurde. Darüber hinaus wird sie auch in vielen anderen Informatik-Zeitschriften verwendet und liegt insbesondere auch der einschlägigen Datenbank *Compuscience* zugrunde.
Die CR Classification besteht aus folgenden Elementen:

- Die **general terms** sind eine vorgegebene Menge von allgemeinen Begriffen, die zur Facettierung dienen.
- Die **classification codes** stellen eine dreistufige monohierarchische Klassifikation dar.

- Innerhalb einer einzelnen Klasse dienen die **subject headings** zur weiteren Untergliederung. Neben der für jede Klasse vorgegebenen Menge von natürlichsprachlichen Bezeichnungen sind auch alle Eigennamen als subject headings erlaubt.
- Schließlich können jedem Dokument noch **free terms** als zusätzliche, frei wählbare Stichwörter zugeordnet werden.

4.3.2.3.1 General terms:

Die general terms der CR Klassifikation sind in Tabelle 4.2 aufgelistet.

ALGORITHMS	MANAGEMENT
DESIGN	MEASUREMENT
DOCUMENTATION	PERFORMANCE
ECONOMICS	RELIABILITY
EXPERIMENTATION	SECURITY
HUMAN FACTORS	STANDARDIZATION
LANGUAGES	THEORY
LEGAL ASPECTS	VERIFICATION

Tabelle 4.2: General terms der CR Klassifikation

4.3.2.3.2 Klassen und subject headings

Die Hauptklassen der CR Klassifikation sind folgende:
A. GENERAL LITERATURE
B. HARDWARE
C. COMPUTER SYSTEMS ORGANIZATION
D. SOFTWARE
E. DATA
F. THEORY OF COMPUTATION
G. MATHEMATICS OF COMPUTING
H. INFORMATION SYSTEMS
I. COMPUTING METHODOLOGIES
J. COMPUTER APPLICATIONS
K. COMPUTING MILIEUX

Am Beispiel der Klasse H.3 zeigen wir die classification codes und die zugehörigen subject headings:

H.3 INFORMATION STORAGE AND RETRIEVAL
- H.3.0 General
- H.3.1 Content Analysis and Indexing
  - Abstracting methods
  - Dictionaries
  - Indexing methods
  - Linguistic processing
  - Thesauruses
- H.3.2 Information Storage
  - File organization
  - Record classification
- H.3.3 Information Search and Retrieval
- H.3.2 Information Storage
  - Clustering
  - Query formulation
  - Retrieval models
  - Search process
  - Selection process
- H.3.4 System and Software
  - Current awareness systems
  - (selective dissemination of information-SDI)
  - Information networks
  - Question-answering (fact retrieval) systems
- H.3.5 Online Information Services
  - Data bank sharing
- H.3.6 Library Automation
  - Large text archives
- H.3.m Miscellaneous

4.3.3 Thesauri

Nach DIN 1463 ist ein Thesaurus eine geordnete Zusammenstellung von Begriffen mit ihren (natürlichsprachlichen) Bezeichnungen. Die wesentlichen Merkmale eines Thesaurus sind folgende:

a) terminologische Kontrolle durch
   - Erfassung von Synonymen,
   - Kennzeichnung von Homographen und Polysemen,
   - Festlegung von Vorzugsbenennungen,

b) Darstellung von Beziehungen zwischen Begriffen.

4.3.3.1 Terminologische Kontrolle

Die terminologische Kontrolle soll zur Reduktion von Mehrdeutigkeiten und Unschärfen der natürlichen Sprache dienen. Hierzu dienen die Synonymkontrolle, die Polysemkontrolle und die Zerlegungskontrolle.

4.3.3.1.1 Synonymkontrolle

Bei der Synonymkontrolle werden Bezeichnungen zu Äquivalenzklassen zusammengefaßt. Man kann folgende Arten von Synonymie unterscheiden:

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

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.

4.3.3.1.2 Polysemkontrolle

Bei der Polysemkontrollewerden mehrdeutige Bezeichnungen auf mehrere Äquivalenzklassen aufgeteilt. Man kann hierbei noch zwischen Homographen *(Bs. Tenor)* und eigentlichen Polysemen *(Bs. Bank)* unterscheiden.

4.3.3.1.3 Zerlegungskontrolle

Bei der Zerlegungskontrolle ist die Frage zu beantworten, wie spezifisch einzelne Begriffe im Thesaurus sein sollen. Gerade im Deutschen mit seiner starken Tendenz zur Kompositabildung *(Bs. Donaudampfschiffahrtsgesellschaftskapitän)* ist die Bildung zu spezieller Begriffe eine große Gefahr. Diese **Präkoordination** führt zu folgenden Nachteilen:

- Der Thesaurus wird zu umfangreich und unübersichtlich.
- Zu einer Äquivalenzklasse gibt es keine oder nur wenige Dokumente in der Datenbank

Den entgegengesetzten Ansatz verfolgt das UNITERM-Verfahren: Hierbei werden nur solche Begriffe (Uniterms) in den Thesaurus aufgenommen, die nicht weiter zerlegbar sind. Zur Wiedergabe eines Sachverhaltes müssen dann mehrere Uniterms verkettet werden. Diese sogenannte **Postkoordination** führt aber zu größerer Unschärfe beim Retrieval *(Beispiel: Baum + Stamm = Baumstamm / Stammbaum)*.
Bei der Thesaurusmethode versucht man, durch einen Kompromiß zwischen beiden Ansätzen deren Nachteile zu vermeiden.

4.3.3.2 Äquivalenzklasse — Deskriptor

Die terminologische Kontrolle liefert Äquivalenzklassen von Bezeichnungen. Diese können auf zwei verschiedene Arten dargestellt werden:

1. In einem **Thesaurus ohne Vorzugsbenennung** werden alle Elemente der Äquivalenzklasse gleich behandelt, d.h., jedes Element steht für die Äquivalenzklasse. Diese Vorgehensweise wird wegen des erhöhten Aufwands selten angewandt.
2. Bei einem **Thesaurus mit Vorzugsbenennung** wird ein Element der Äquivalenzklasse zur Benennung ausgewählt. Dieses Element bezeichnet man dann als **Deskriptor**.

Im folgenden betrachten wir nur Thesauri mit Vorzugsbenennung.

4.3.3.3 Beziehungsgefüge des Thesaurus

Neben der terminologischen Kontrolle ist die Darstellung von Beziehungen zwischen Begriffen die zweite Hauptaufgabe eines Thesaurus. Dabei werden verschiedene Arten von Beziehungen unterschieden.

4.3.3.3.1 Äquivalenzrelation

Äquivalenzrelationen verweisen von Nicht-Deskriptoren auf Deskriptoren. Sie werden meist bezeichnet als „**Benutze Synonym**“ (BS) oder im Englischen als USE-Relation. Die Umkehrrelation bezeichnet man als „**Benutzt für**“ (BF, im Englischen “used for” (UF)). Beispiele hierfür sind:
(*Fernsprecher* **BS** *Telefon und Telefon* **BF** *Fernsprecher*

4.3.3.3.2 Hierarchische Relation

Hierarchische Relationen verbinden jeweils zwei Deskriptoren. Man bezeichnet sie als „**Unterbegriff**“ (UB) bzw. „**Oberbegriff**“ (OB), im Englischen “narrower term” (NT) und “broader term” (BT). Beispiele:
Obstbaum **UB** *Steinobstbaum und Steinobstbaum* **OB** *Obstbaum*

4.3.3.3.3 Assoziationsrelation

Die Assoziationsrelation verweist von einem Deskriptor auf einen begriffsverwandten anderen Deskriptor. Im Gegensatz zu den beiden anderen Relationen ist die Assoziationsrelation symmetrisch. Man bezeichnet sie als „**verwandter Begriff**“ (VB, im Englischen “see also” (SEE)). Beispiele:
Obstbaum **VB** *Obst und Obst* **VB** *Obstbaum*

4.3.3.4 Darstellung des Thesaurus

4.3.3.4.1 Deskriptor-Einträge

Ein Deskriptor-Eintrag in einem Thesaurus enthält neben der Vorzugsbenennung häufig noch mehrere der folgenden Angaben:

- Begriffsnummer,
- Notation / Deskriptor-Klassifikation,
- Scope note / Definition,
- Synonyme,
- Oberbegriffe / Unterbegriffe,
- Verwandte Begriffe,
- Einführungs-/Streichungsdatum.

Abbildung 4.7 zeigt ein Beispiel für einen Ausschnitt aus einem Thesaurus.

0.0058
Magnetband
VB Magnetbandlaufwerk

0,0045
Magnetbandgerät
BS Magnetbandlaufwerk NE7

0. 0046
Magnetbandkassette
NO NE83
BF Kassette
BF MB-Kassette
OB Datenträger
VB Magnetbandkassettenlaufwerk

0.0051
Magnetbandkassettengerät
BS Magnetbandkassettenlaufwerk NE7

0.0050
Magnetbandkassettenlaufwerk
NO NE7
BF Magnetbandkassettengerät
BF MB-Kassettengerät
OB Datenausgabegrät
OB Dateneingabegerät
OB Datenspeichertechnik
VB Magnetbandkassette

0.0044
Magnetbandlaufwerk
NO NE7
BF Magnetbandgerät
OB Bandgerät
OB Datenausgabegerät
OB Dateneingabegerät
OB Datenspeichertechnik
VB Magnetband

0.0059
Magnetfeld
NO WD2
OB Magnetismus

0.0060
Magnetismus
NO WD2
BF Barkhausen-Effek
BF Ferromagnetismus

Magnetismus (Forts.)
BF Halleffekt
BF Induktion
OB Elektrodynamik
UB Magnetfeld
BIK Geophysik
BFK Erdmagnetismus
BIK Optik
BFK Faraday-Effekt

0.0070
Magnetkarte
NO NE87
BF Telefonkärtchen
OB Datenträger
VB Kartensystem

0.0073
Magnetkartensystem
NO ECS
OB Kartensystem

0.0074
Magnetkartentelefon
NO GK72
BF Makatel
OB Kartentelefon

0 0077
Magnetplatte
NO NE82
OB Datenspeicher
OB Datenträger
VB Magnetplattenlaufwerk
BIK Datenspeicher
BFK Plattenspeicher

0.0081
Magnetplattengerät
BS Magnetplattenlaufwerk NE7

0.0079
Magnetplattenlaufwerk
NO NE7
BF Magnetplattengerät
OB Datenausgabegerät
OB Dateneingabegerät
OB Datenspeichertechnik
VB Magnetplatte

Abbildung 4.7: Auszug aus einem Thesaurus

4.3.3.4.2 Gesamtstruktur des Thesaurus

Bei einem IR-System, das zur Recherche in einer Datenbasis mit Thesaurus verwendet wird, sollte auch der Thesaurus zugreifbar sein, wobei spezielle Funktionen zum Suchen im Thesaurus und mit Hilfe des Thesaurus angeboten werden sollten (z.B. wahlweise Einbeziehen von allen Unter-/Oberbegriffen). Daneben ist der Thesaurus aber meistens auch in gedruckter Form verfügbar. Der **Hauptteil** eines Thesaurus enthält dabei die Deskriptor-Einträge, die entweder alphabetisch oder systematisch geordnet sind. Darüber hinaus enthält ein Thesaurus in der Regel noch zusätzliche Register mit Verweisen auf die Deskriptor-Einträge:

- komplementär zum Hauptteil eine systematische bzw. alphabetische Auflistung der Deskriptoren,
- für mehrgliedriger Bezeichnungen einen speziellen Index für deren Komponenten:
  - KWIC — keyword in context

    computer *system*
    storage *system*
    system *analysis*
    system *design*
  - KWOC — keyword out of context

    system:
    computer ...
    storage ...
    ... *analysis*
    ... *design*

4.3.3.5 Thesauruspflege

Da ein Anwendungsgebiet nie statisch ist und man darüber hinaus auch nicht annehmen kann, daß die erste Version eines Thesaurus bereits alle Ansprüche erfüllt, ist eine ständige Pflege des Thesaurus' notwendig. Insbesondere erfordern folgende Faktoren eine laufende Anpassung des Thesaurus':

- Entwicklung des Fachgebietes,
- Entwicklung der Fachsprache,
- Analyse des Indexierungsverhaltens und der Indexierungsergebnisse,
- Beobachtung des Benutzerverhaltens,
- Analyse der Rechercheergebnisse.

Bei den daraus resultierenden Änderungen muß darauf geachtet werden, daß die Konsistenz des Thesaurus' erhalten bleibt.

4.3.4 Semantische Netze am Beispiel KL-ONE

KL-ONE ist ein im Bereich der KI (Künstliche Intelligenz) entwickelter Formalismus für semantische Netzwerke [Brachman/Schmolze 85]. Im Vergleich zu den in der Datenbankforschung entwickelten Datenmodellen bietet KL-ONE wesentlich ausdrucksstärkere Mittel zur Definition von Begriffstaxonomien. Durch den Trend zu objektorientierten Datenbanksystemen ist KL-ONE in jüngster Zeit für die Datenbankforschung zunehmend wichtiger geworden (siehe z.B. [Heuer/Scholl 91]).
Bei KL-ONE besteht ein semantisches Netzwerk aus 2 Teilen, die eng miteinander verbunden sind:

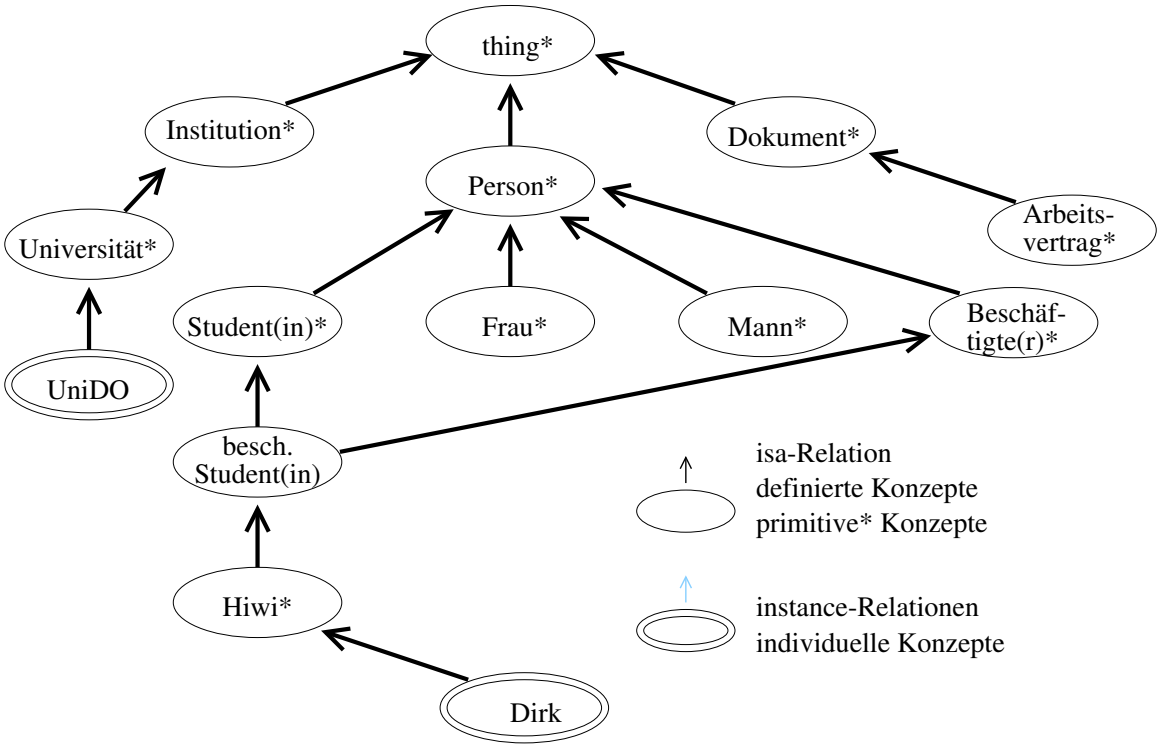
1. TBox (terminological box) zur Definition der Konzepte. Die TBox ist mit dem Schema einer Datenbank vergleichbar.
2. ABox (assentional box) enthält Aussagen über konkrete Instanzen, entspricht also dem Zustand einer DB.

Die wichtigsten Elemente eines semantischen Netzwerks in KL-ONE sind Konzepte (vergleichbar den Objekttypen im ERM) und Rollen ($\hat{=}$ zweistellige Beziehungstypen im ERM), wobei erstere den Knoten und letztere den Kanten eines Graphen entsprechen. Darüber hinaus gibt es noch weitere Beziehungen, sowohl zwischen Konzepten als auch zwischen Rollen, die in erster Linie der Darstellung der Begriffshierarchie dienen.

Konzepte
Es werden drei verschiedene Arten von Konzepten unterschieden:

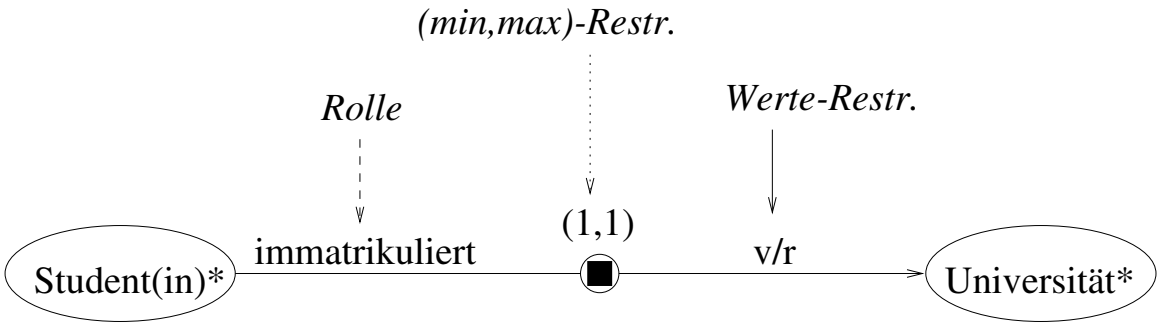
- Primitive Konzepte sind Konzepte, bei denen nur notwendige, aber nicht hinreichende Bedingungen zur Definition eines Konzeptes angegeben wurden (z.B. Universität definiert als Spezialfall einer Institution). Bei praktischen Anwendungen zeigt sich, daß für die Mehrzahl der Begriffe keine hinreichenden Bedingungen angegeben werden können, womit die meisten Begriffe also primitiv sind. Unter den primitiven Konzepten sind die „root concepts“ besonders ausgezeichnet, für die keine Oberbegriffe mehr angegeben sind (z.B. das Konzept „thing“).
- Bei definierten Konzepten sind die notwendigen und hinreichenden Bedingungen zur Definition des Konzepts angegeben.
- Individuelle Konzepte stellen Instanziierungen von primitiven oder definierten Konzepten dar.

Die TBox enthält nur definierte und primitive Konzepte, individuelle Konzepte kommen nur in der ABox vor. Bei der Darstellung werden die Bezeichnungen primitiver Konzepte durch einen * markiert, und individuelle Konzepte werden durch eine doppelte Umrandung gekennzeichnet.
Die Beziehung zwischen Unter- und Oberbegriffen in KL-ONE wird durch die isa-Relation angegeben. Dabei können insbesondere mehrere Oberbegriffe zu einem Konzept angegeben werden ($\rightarrow$ Vererbung).



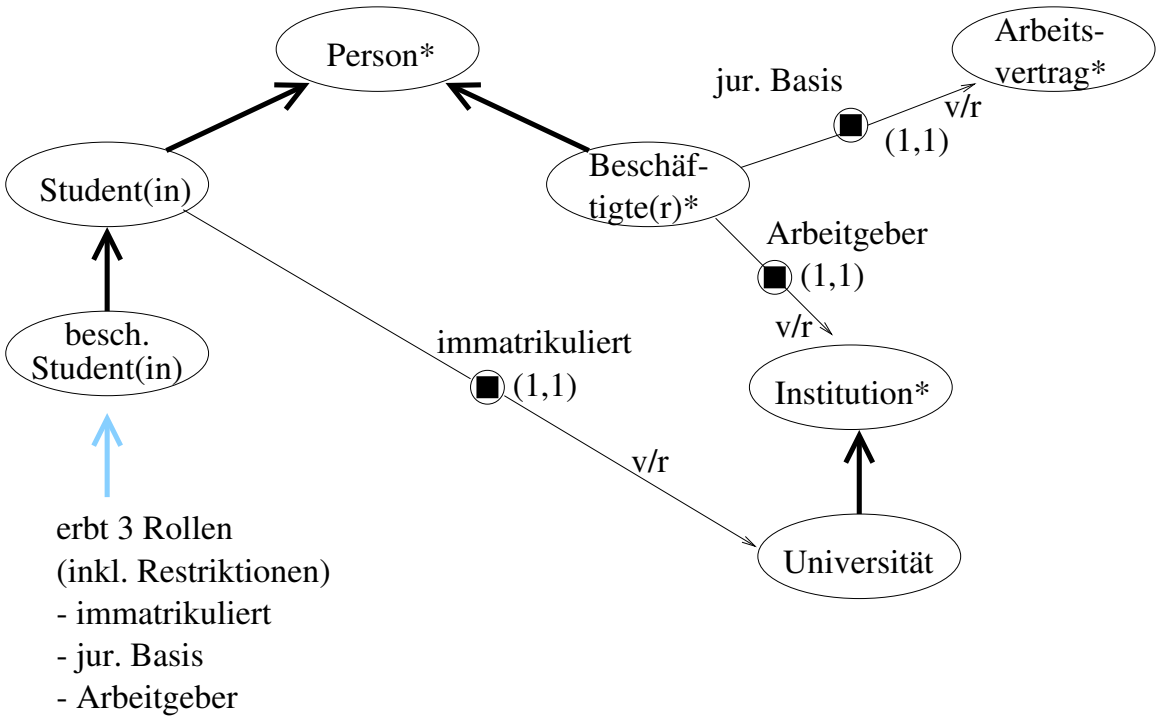
Rollen

Rollen stellen gerichtete, benannte Kanten dar, die von einem „domain concept“ ausgehen und zu einem „range concept“ hin (z.B. zeigt die Rolle „immatrikuliert“ vom domain concept „Student(in)“ auf das range concept „Universität“). Zusätzlich kann eine Kardinalität (number restriction) als Intervall angegeben werden. Fehlt die Angabe einer Kardinalität, so entspricht dies der Angabe $(0, \infty)$.



Bei der Unter-/Oberbegriffsbeziehung vererbt der Oberbegriff alle Rollen mit den angegebenen Einschränkungen bezüglich number restriction und range concept an den Unterbegriff.

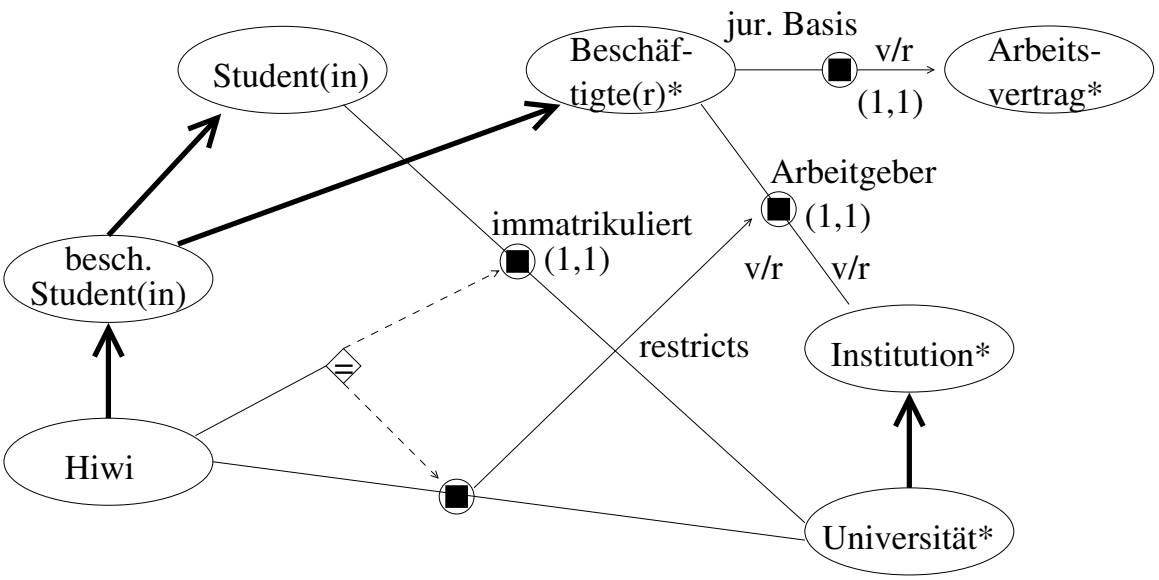
Vererbung



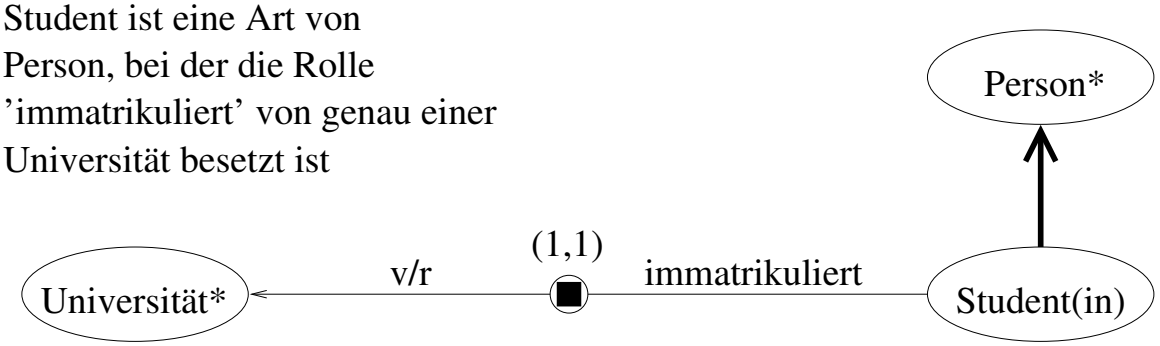
Um einen Unterbegriff aus einem Oberbegriff auszudifferenzieren, gibt es folgende Möglichkeiten:

1. durch polyhierarchische Einordnung, also die Angabe von mehr als einem Oberbegriff (s. z.B. besch. Student(in))
2. durch die Angabe zusätzlicher Rollen (s. z.B. Student vs. Person)
3. durch Restriktionen auf vorhandenen Rollen (sog. „restricts“-Beziehung zwischen Rollen):
   a) durch Einschränkung der Kardinalität,
   b) durch Einschränkung des range concepts (sog. „value restriction“) einer Rolle,
   c) durch „role value map“: Dabei wird für zwei Rollen desselben Konzeptes eine bestimmte Beziehung für die Mengen der Filler der beiden Rollen spezifiziert (so ist z.B. „Hiwi“ ein „beschäftigte(r) Student(in)“, der an derselben Hochschule beschäftigt ist, an der er immatrikuliert ist).
   d) durch Differenzierung von Rollen („diffs“), z.B. Differenzierung der Rolle „Name“ in die beiden Rollen „Vorname“ und „Nachname“. Im Prinzip wird dadurch eine Hierarchie über Rollen definiert. Die Differenzierung von Rollen wird im folgenden nicht weiter betrachtet.

Meist werden mehrere dieser Möglichkeiten kombiniert zur Definition eines neuen Begriffs verwendet. Zusätzlich muß in jedem Fall angegeben werden, ob das so gebildete Konzept primitiv oder definiert ist.



Student ist eine Art von
Person, bei der die Rolle
'immatrikuliert' von genau einer
Universität besetzt ist



Subsumption

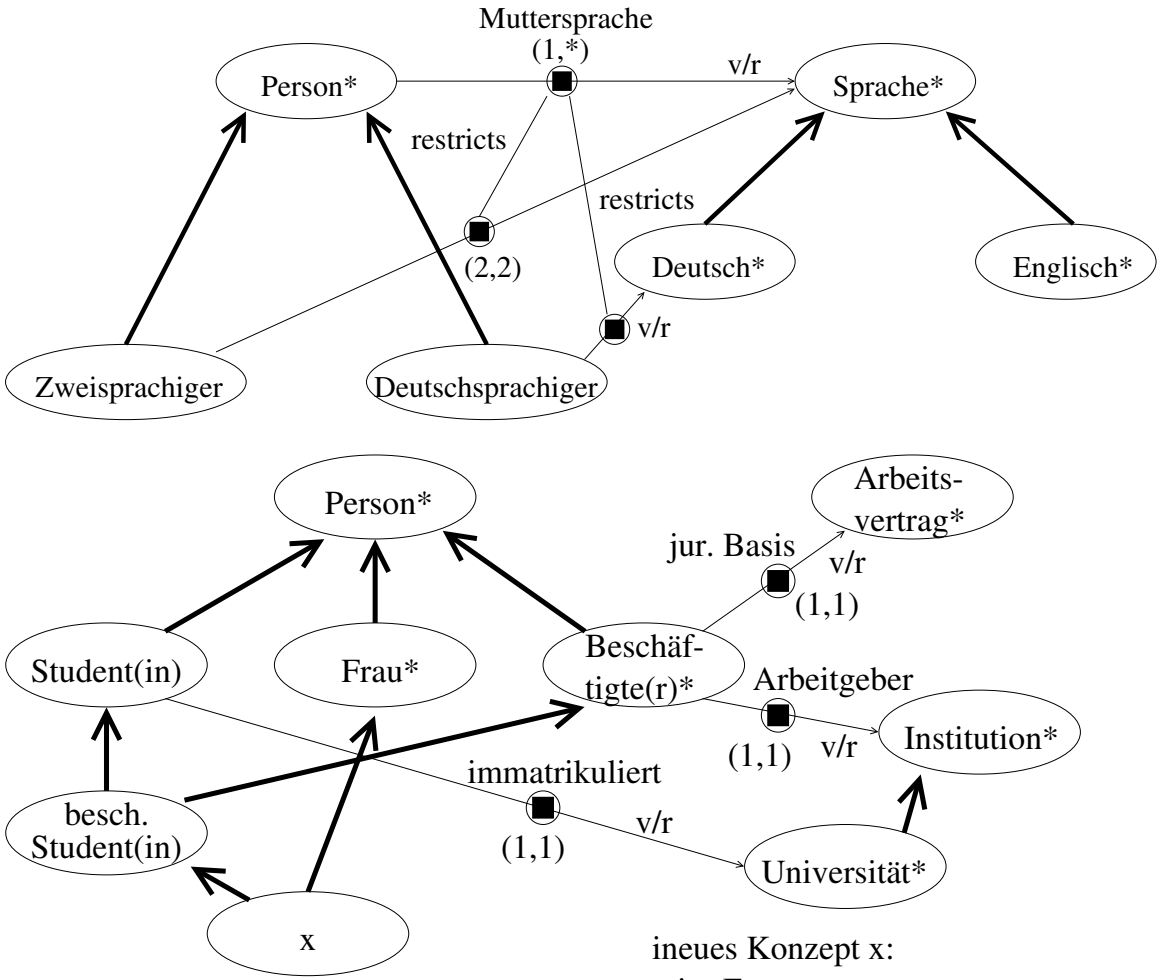
Durch die vielfältigen Möglichkeiten zur Definition von Begriffen stellt sich das Problem, wann ein Begriff A Oberbegriff von einem Begriff B ist, ihn also subsumiert. Eine präzise Definition der Subsumptionsbeziehung ist insbesondere für umfangreichere Netzwerke notwendig. Darauf aufbauend kann ein Klassifikator implementiert werden, der neu definierte Begriffe automatisch einordnet (wie im nachfolgenden Beispiel).

Für Konzepte gilt: C_1 subsumiert $C_2 \Leftrightarrow$

1. Alle primitiven Oberbegriffe von C1 (bzw. C1 selbst, falls primitives Konzept) sind auch Oberbegriffe von C2,
2. jede Rolle von C1 subsumiert die gleichnamige Rolle von C2,
3. alle role value maps von C2 werden von C1 subsumiert.

Für die Rollen-Subsumption gilt: Die Rolle r_1 subsumiert die Rolle $r_2 \Leftrightarrow$

1. Die Kardinalität von r_1 subsumiert die Kardinalität von r_2, das heißt, die number restriction von r_2 ist identisch mit oder ein Teilintervall der number restric-

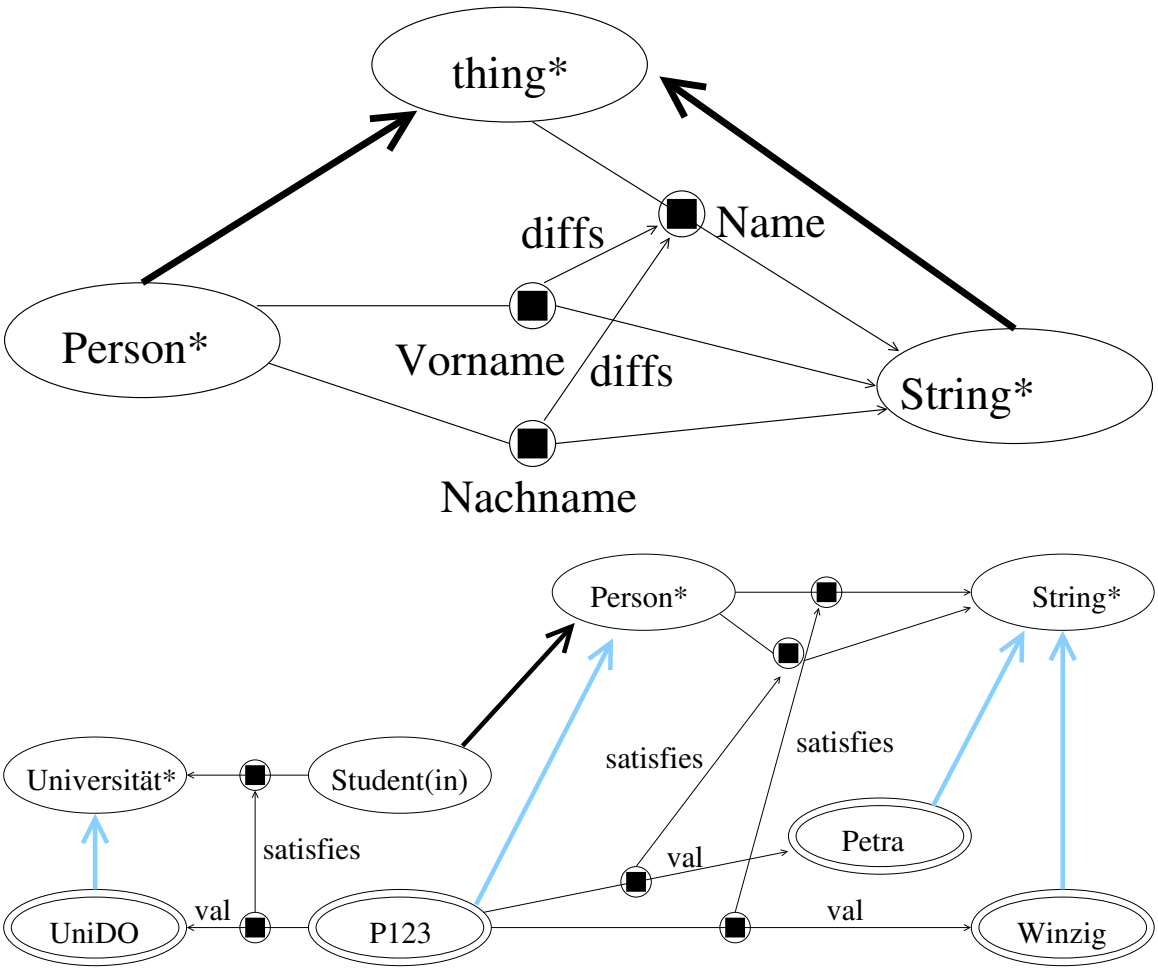


tion von r_1.

2. Das range concept von r_1, subsumiert das range concept von r_2.

Sei $RVM_i = (rel_i, r_1, r_2)$ die role value map eines Konzeptes C_i, wobei r_1 und r_2 die beteiligten Rollen bezeichnen und rel_i die Relation zwischen den beiden Rollenfüllern. Dann gilt: Ein role value map $RVM_2 = (rel_2, r_1, r_2)$ eines Konzeptes C_2 wird von einem Konzept C_1 subsumiert $\Leftrightarrow$

- nicht beide Rollen r_1 und r_2 bei C_1 existieren, oder
- die beiden Rollen bei C_1 existieren, aber keine role value map zwischen den beiden Rollen angegeben ist, oder
- beide Rollen existieren und es ist $RVM1 = (rel_1, r_1, r_2)$ definiert, dann muß $rel_1 rel_2$ subsumieren.



Auf der Basis dieser Definition kann die Subsumption von zwei Konzepten C_1 und C_2 und insbesondere auch deren Äquivalenz festgestellt werden, d.h. C_1 subsumiert C_2 und umgekehrt.

ABox

Zur Definition von individuellen Konzepten in der ABox gibt es folgende Beziehungstypen in KL-ONE:

- „instance"-Relation zur Zuordnung eines individuellen Konzeptes zum zugehörigen (primitiven oder definierten) Konzept der TBox,
- „satisfies"- und „val"-Beziehung zur Angabe der Rollenfüller eines individuellen Konzeptes, wobei „val" auf das individuelle Konzept zeigt, das den aktuellen Wert der bei „satisfies" angegebenen Rolle darstellt.

In Weiterentwicklungen von KL-ONE wie z.B. KRYPTON [Brachman et al. 83, Brachman/Levesque 86] gibt es zusätzlich die Möglichkeit, beliebige prädikatenlogische Formeln (1. Stufe) in der ABox anzugeben (z.B. kann dann das System eine

Person als Student erkennen, von der man nur weiß, daß sie entweder an der Uni Bochum oder an der Uni Dortmund eingeschrieben ist)

4.3.5 Dokumentationssprachen vs. Freitext

Beim Vergleich mit der Freitextsuche sind folgende Vor- und Nachteile von Dokumentationssprachen zu nennen:

+ Durch die Abbildung verschiedener Textformulierungen auf eine einzige Bezeichnung kann ein höherer Recall erreicht werden.
+ Da das kontrollierte Vokabular keine mehrdeutigen Begriffe zuläßt, kann auch eine höhere Precision erreicht werden.
+ Da ein Benutzer ein gesuchtes Konzept nur auf die entsprechende Benennung in der Dokumentationssprache abbilden muß, ergibt sich eine größere Benutzerfreundlichkeit.
– Die Benutzung des Systems setzt die Kenntnis der Dokumentationssprache voraus; für gelegentliche Benutzer ist diese Hürde zu hoch.
– Aufgrund der i.a. groben Granularität des kontrollierten Vokabulars kann bei spezifischen Anfragen die Precision im Vergleich zur Freitextsuche sinken.
– Bei der Eingabe neuer Dokumente in die Datenbasis erhöht sich der Erschließungsaufwand deutlich, weil die Klassifikation bzw. Indexierung meist manuell erfolgt. Allerdings verringert sich durch diese Maßnahme der Aufwand bei den Recherchen, so daß die Gesamtbilanz wohl eher positiv ist.

Um die Nachteile des kontrollierten Vokabulars bei der Recherche zu kompensieren, kombinieren heutige kommerziell angebotenen Datenbasen beide Suchmöglichkeiten, so daß die Dokumentationssprache die Freitextsuche ergänzt.

4.4 Beurteilung der Verfahren zur Repräsentation von Textinhalten

- Obwohl rein intuitiv die Vorteile von Dokumentationssprachen überzeugen, ist deren Nutzen jedoch wissenschaftlich sehr umstritten. Der Grund hierfür ist die unzureichende experimentelle Basis für diesen Vergleich. Seit den Anfang der 60er Jahre von Cyril Cleverdon geleiteten Cranfield-Experimenten [Cleverdon 91], wo alle Dokumentationssprachen deutlich schlechter abschnitten als eine Freitextsuche mit Terms in Stammform, neigt die Mehrzahl der IR-Forscher zu der Ansicht, daß Dokumentationssprachen überflüssig sind. Allerdings wurden die damaligen Experimente mit nur 1400 Dokumenten durchgeführt, so daß die Gültigkeit der Resultate für heutige Datenbasen in der Größenordnung von 10^6 Dokumenten mit Recht bezweifelt werden muß. Auch einige wenige neuere Vergleiche [Salton 86] lassen keine endgültige Aussage zu dieser Problematik zu.

- Im Rahmen der TREC-Initiative werden verschiedene IR-Verfahren auf eine Datenbasis mit 2 GB Text angewendet und die Ergebnisse miteinander verglichen. Die auf der ersten TREC-Konferenz [Harman 93] präsentierten Ergebnisse zeigen, daß halb-formale Konzepte (wie z.B. geographische oder Datumsangaben) durch eine reine Freitextsuche nicht abzudecken sind, so daß zumindest für diesem Bereich Dokumentationssprachen notwendig sind.
- Es liegt nahe, nach dem Einsatz von wissensbasierten Verfahren im IR zu fragen. Wie auch die Studie [Krause 92] zeigt, gibt es derzeit noch keine erfolgversprechenden Ansätze in diesem Bereich. Das Hauptproblem ist das Fehlen ausreichend großer Wissensbasen. Hier versucht das in [Lenat et al. 90] beschriebene CYC-Projekt anzusetzen, dessen Ziel es ist, eine ganze Enzyklopädie als Wissensbasis bereitzustellen.
- Syntaktische Verfahren sind wohl hauptsächlich für die Identifikation von Nominalphrasen einsetzbar.
- Maschinenlesbare Wörterbücher sind in immer größerem Maße verfügbar. Sie unterstützen die morphologische Analyse bei stark flektierten Sprachen und die Erkennung von Nominalphrasen. Einige Forschungsgruppen untersuchen auch deren Einsatz für die Disambiguierung von Begriffen.

4.5 Zusammenhang zwischen Modellen und Repräsentationen

4.5.1 Textrepräsentation für IR-Modelle

Abschließend zu diesem Kapitel soll eine Einordnung der verschiedenen vorgestellten Ansätze zur Repräsentation von Textinhalten im Hinblick auf ihre Kombination mit IR-Modellen versucht werden.

4.5.2 Einfache statistische Modelle

Zunächst illustrieren wir die Vorgehensweise bei der Freitextindexierung an einem Beispieltext:
Experiments with Indexing Methods.
The analysis of 25 indexing algorithms has not produced consistent retrieval performance. The best indexing technique for retrieving documents is not known.
Zunächst werden die (oben unterstrichenen) Stoppwörter entfernt:
experiments indexing methods analysis indexing algorithms produced consistent retrieval performance best indexing technique retrieving documents known.
Die anschließende Stammformreduktion liefert folgendes Ergebnis:

experiment index method analys index algorithm produc consistent retriev perform best index techni retriev document.
Die einfachsten IR-Modelle betrachten Dokumente als Mengen von Terms, so daß die zugehörige Repräsentation eines Dokumentes wie folgt aussieht:
algorithm analys best consistent document experiment index method perform produc retriev techni.
Wir nehmen nun an, daß wir ein Dokument durch einen Beschreibungsvektor $\vec{x} = (x_1, \ldots, x_n)$ repräsentieren, wobei die Komponente x_i jeweils das Vorkommen des Terms $t_i \in T = \{t_1, \ldots, t_n\}$ in dem aktuellen Dokument beschreibt.
Im Falle einer **Term-Menge** sind die Vektor-Komponenten binär, also $x_i = 1$, falls t_i im Dokument vorkommt, und $x_i = 0$ sonst.
Als eine Verbesserung dieser Repräsentationsform kann man die Vorkommenshäufigkeit des Terms im Dokument berücksichtigen. Somit haben wir jetzt eine **Multi-Menge von Terms**, repräsentiert durch $x_i \in \{0, 1, 2, \ldots\}$.
Die semantische Sicht auf Texte besteht hier also aus dieser Multimenge von Terms. Die eigentliche Semantik (z.B. die Unterscheidung zwischen wichtigen und unwichtigen Wörtern) kommt jedoch durch das auf diese Sicht aufbauende Retrievalmodell zustande, und zwar bei der Abbildung auf die Objektattribute mit Hilfe von statistischen Verfahren!

4.5.3 Vorkommensformen

Eine weiter differenzierte Repräsentation (semantische Sicht) liefert das im Zusammenhang mit dem Darmstädter Indexierungsansatz [Lustig 86] entwickelte Konzept der **Vorkommensformen**. Diese vereinigt sowohl die **Signifikanz** eines Terms als auch seine **Identifikationssicherheit** bezüglich eines Dokumentes. Die Vorkommensform wird dabei zunächst durch eine Menge formaler Parameter charakterisiert. Die Signifikanz kann durch die Vorkommenshäufigkeit und den Vorkommensort (z.B. Titel vs. Abstract) charakterisiert werden. Die Identifikationssicherheit hängt von der Wortart (Einzelwort vs. Nominalphrase) ab:

- Bei Einzelwörter können wir unterscheiden, ob Gleichheit zwischen Frageterm und Dokumentterm in der Grundform oder nur in der Stammform besteht:
  Suche nach „computer“
  Dokument d_1: ... computers ...
  Dokument d_2: ... computing ...
  In der Regel ist die Gleichheit in der Grundform sicherer als in der Stammform.
- Bei Nominalphrasen wurden in Abschnitt 4.2.2.3.2 bereits mehrere Kriterien genannt, von denen die Identifikationssicherheit abhängt.
  Suche nach „text retrieval“
  ... retrieval from texts ...
  ... text and fact retrieval ...

... text is stored and retrieved ...
... this text is about fact retrieval ...
Hier kann man z.B. die Identifikationssicherheit als Funktion des Abstandes betrachten. Werden syntaktische Verfahren eingesetzt, dann kann stattdessen die Struktur des Parsing-Baumes betrachtet werden.

Basierend auf diesen Parametern kann nun eine Menge von Vorkommensformen $V = \{v_1, \ldots, v_L\}$ definiert werden, indem für jede Vorkommensform jeweils bestimmte Bedingungen bezüglich der einzelnen Parameterwerte festgelegt werden (z.B. v_1 als: Einzelwort in Grundform mit mindestens zwei Vorkommen, davon einmal im Titel). Das Vektorelement x_i des Beschreibungsvektors gibt dann die Vorkommensform $v \in V$ von t_i im Dokument an.
Ein besserer Weg ist aber die Verwendung von probabilistischen, beschreibungsorientierten Indexierungs- und Retrievalmodellen (siehe 6.3.3 und 6.3.1.2.1), die mächtig genug sind, um alle Daten der Repräsentation auszunutzen und in probabilistische Gewichte abzubilden.

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.

Abbildung 4.8: Beispieldokument aus der PHYS-Datenbasis

Die bisher in diesem Abschnitt diskutierten Repräsentationen basieren alle auf dem Freitext der Dokumente. Es ist aber ebenfalls möglich, Texte durch eine Dokumentationssprache zu repräsentieren. Wir betrachten hier nur den Fall einer Deskribierung durch eine Menge von Elementen der Dokumentationssprache, die wir der Einfachheit halber als Deskriptoren bezeichnen (Klassifikationen wären ebenso möglich). Abgesehen von dem Fall einer intellektuell erstellten Deskribierung ist aber auch eine **automatische Indexierung** möglich. Hierzu muß ein sogenanntes Indexierungswörterbuch vorhanden sein, das Paare von Textterms und zugehörigen Deskriptoren enthält. Diese Paare können zusätzliche Attribute besitzen, z.B. den Wert der bedingten Wahrscheinlichkeit $P(s|t)$ daß der Deskriptor s zuzuteilen ist, wenn der Term t im Text auftritt (eine solche Wahrscheinlichkeit kann aus bereits vorliegendem intellektuell indexierten Material geschätzt werden). Abbildung 4.8 zeigt ein Dokument aus dem in [Lustig 86] beschriebenen System AIR/PHYS zur automatischen Indexierung von Physik-Dokumenten, das seit 1985 beim Fachinformationszentrum Karlsruhe praktisch eingesetzt wird. Für den Deskriptor TEMPERATURE

DEPENDENCE enthält das zugrundeliegende Indexierungswörterbuch u.a. Paare mit den Termen TEMPERATURE, HALL, ELECTRICAL COEFFICIENT, HALL COEFFICIENT und MEASURE ELECTRICAL. Dieses Beispiel demonstriert, daß die Beziehung zwischen Dokument und Deskriptor relativ komplex werden kann. Die sogenannte „Relevanzbeschreibung“ enthält nun Angaben über den Deskriptor, das Dokument, die Beziehungen zwischen den Textterms und dem Deskriptor sowie über die Vorkommensformen der Terme. Es ist offensichtlich, daß all diese Angaben nicht mehr auf einen einzigen Wert im Beschreibungsvektor abgebildet werden können. Für diesen Fall müssen die oben erwähnten beschreibungsorientierten Modelle angewendet werden.

So wie die Repräsentation für Dokumente erstellt wird, können im Prinzip auch Fragen, die in natürlicher Sprache vorliegen, repräsentiert werden. Anschließend kann auf die so erstellten Repräsentationen eine Retrievalfunktion zum Vergleich von Dokument und Frage angewendet werden, die dann als Resultat ein Retrievalgewicht für das Frage-Dokument-Paar liefert.

Kapitel 5

Nicht-probabilistische IR-Modelle

5.1 Notationen

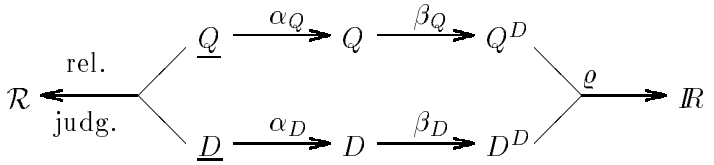


Abbildung 5.1: Konzeptionelles Modell für Textretrieval

Als grundlegendes konzeptionelles Modell für alle Arten von Modellen für (Text-)Retrieval verwenden wir das in Abb. 5.1 dargestellte Modell, das eine Vereinfachung des in Abschnitt 2.4 vorgestellten allgemeinen Modells ist. Dabei steht $\underline{D}$ für die Menge der Dokumente in der Datenbasis und $\underline{Q}$ für die Menge der Anfragen an das IRS. Zwischen den Dokumenten und den Anfragen besteht die Relevanzbeziehung, die hier als Abbildung in die Menge $\mathcal{R}$ der möglichen Relevanzurteile aufgefaßt wird. Die in dem IRS repräsentierte semantische Sicht von Dokumenten bezeichnen wir im folgenden einfach als Dokumentrepräsentationen D, und die formalisierten Anfragen als Frage-Repräsentationen Q. Diese entstehen aus den ursprünglichen Objekten durch die Abbildungen α_D und α_Q. 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. Die Erstellung der Beschreibungen aus den Repräsentationen und die (mehr oder

weniger begründete) Definition einer Retrievalfunktion hängt von dem jeweils zugrundegelegten Retrievalmodell ab. In diesem und dem folgenden Kapitel werden verschiedene solcher Retrievalmodelle beschrieben, die nicht nur in der Retrievalfunktion, sondern auch schon bzgl. der zugrundegelegten Repräsentationen und den daraus abgeleiteten Beschreibungen differieren.
Nachstehend verwenden wir außerdem folgende Abkürzungen:

$T = \{t_1, \ldots, t_n\}$: Indexierungsvokabular
$\underline{q_k}$: Frage
q_k: Frage-Repräsentation (*formalisierte Anfrage*)
q_k^D: Frage-Beschreibung (*Fragelogik*)
$\underline{d_m}$: Dokument
d_m: Dokument-Repräsentation (*semantische Sicht*)
d_m^D: Dokument-Beschreibung (*Objektattribute*)
$\vec{d_m} = \{d_{m_1}, \ldots, d_{m_n}\}$: Dokument-Beschreibung als Menge von Indexierungsgewichten.

5.2 Überblick über die Modelle

	Bool.	Fuzzy	Vektor	Prob.	Cluster.
theoretische Basis:					
– boolesche Logik	x				
– Fuzzy-Logik		x			
– Vektoralgebra			x		x
– Wahrsch.-Theorie				x	
Bezug zur Retrievalqual.		(x)		x	
gewichtete Indexierung		x	x	x	x
gewichtete Frageterms		(x)	x	x	
Fragestruktur:					
– linear			x	x	
– boolesch	x	x	(x)	(x)	
Suchmodus:					
– Suchen	x	x	x	x	
– Browsen					x

Abbildung 5.2: IR-Modelle

Abbildung 5.2 gibt eine Einordnung der hier und im folgenden Kapitel behandelten IR-Modelle. Eingeklammerte Markierungen bedeuten dabei, daß dieses Merkmal im Prinzip zutrifft, diese Variante des Modells allerdings hier nicht behandelt wird.

5.3 Boolesches Retrieval

Boolesches Retrieval ist historisch als erstes Retrievalmodell entwickelt und eingesetzt worden. Vermutlich hat Taube als erster dieses Modell zugrundegelegt, um Retrieval mit Hilfe von Schlitzlochkarten durchzuführen. Auch als man später die Dokumente auf Magnetbändern speicherte, war boolesches Retrieval das einzig anwendbare Modell: aufgrund der geringen Speicherkapazität damaliger Rechner mußte direkt nach dem Einlesen des Dokumentes entschieden werden, ob es als Antwort ausgedruckt werden sollte oder nicht. Obwohl sich die Rechnerhardware seitdem rasant weiterentwickelt hat, hat man in der Praxis dieses Modell bis heute nicht grundlegend in Frage gestellt, sondern sich nur mit einigen funktionalen Erweiterungen begnügt.
Beim booleschen Retrieval sind die Dokumenten-Beschreibungen D^D:
ungewichtete Indexierungen, d.h.

$$d_m^D = \vec{d}_m \quad \text{mit} \quad d_{m_i} \epsilon \{0,1\} \quad \text{für} \quad i = 1, \ldots, n \tag{5.1}$$

Die Frage-Beschreibungen Q^D sind boolesche Ausdrücke, die nach folgenden Regeln gebildet werden:

1. $t_i \epsilon T \Rightarrow t_i \epsilon Q^D$
2. $q_1, q_2 \epsilon Q^D \Rightarrow q_1 \wedge q_2 \epsilon Q^D$
3. $q_1, q_2 \epsilon Q^D \Rightarrow q_1 \vee q_2 \epsilon Q^D$
4. $q \epsilon Q^D \Rightarrow \neg q \epsilon Q^D$

Die Retrievalfunktion ϱ kann man analog zu diesen Regeln ebenso rekursiv definieren:

1. $t_i \epsilon T \Rightarrow \varrho(t_i, \vec{d}_m) = d_{m_i}$
2. $\varrho(q_1 \wedge q_2, \vec{d}_m) = \min(\varrho(q_1, \vec{d}_m), \varrho(q_2, \vec{d}_m))$
3. $\varrho(q_1 \vee q_2, \vec{d}_m) = \max(\varrho(q_1, \vec{d}_m), \varrho(q_2, \vec{d}_m))$
4. $\varrho(\neg q, \vec{d}_m) = 1 - \varrho(q, \vec{d}_m)$

Aufgrund der binären Gewichtung der Terme in der Dokumentbeschreibung kann die Retrievalfunktion ebenfalls nur die Retrievalgewichte 0 und 1 liefern. Daraus resultiert als Antwort auf eine Anfrage eine Zweiteilung der Dokumente der Datenbasis in gefundene ($\varrho = 1$) und nicht gefundene ($\varrho = 0$) Dokumente.
In realen IR-Systemen ist boolesches Retrieval meist nur in einer etwas modifizierten Form implementiert: Gegenüber der Darstellung hier ist die Verwendung der Negation derart eingeschränkt, daß diese nur in Kombination mit der Konjunktion verwendet werden darf, also z.B. in der Form $a \wedge \neg b$; eine Anfrage der Form $\neg b$ oder $a \vee \neg b$ ist hingegen nicht zulässig. Die Gründe für diese Einschränkung sind implementierungstechnischer Art.

5.3.1 Mächtigkeit der booleschen Anfragesprache

Ein wesentlicher (theoretischer) Vorteil der booleschen Anfragesprache besteht in ihrer Mächtigkeit. Man kann zeigen, daß mit einer booleschen Anfrage jede beliebige Teilmenge von Dokumenten aus einer Datenbasis selektiert werden kann. Voraussetzung ist dabei, daß alle Dokumente unterschiedliche Indexierungen (Beschreibungen) besitzen.
Zu einer vorgegebenen Dokumentenmenge $\underline{D}_k \subseteq \underline{D}$ konstruiert man dann die Frageformulierung q_k, die genau diese Dokumente selektiert, wie folgt: Zunächst wird für jedes Dokument eine Frage d_m^Q konstruiert, die nur dieses Dokument selektiert; anschließend werden diese Teilfragen für alle Dokumente $\underline{d}_m \in \underline{D}_k$ disjunktiv miteinander verknüpft.

$$\begin{aligned} d_m^Q &= x_{m_1} \wedge \ldots \wedge x_{m_n} \quad \text{mit} \\ x_{m_i} &= \begin{cases} t_i & \text{falls } d_{m_i} = 1 \\ \neg t_i & \text{sonst} \end{cases} \\ q_k &= \bigvee_{\underline{d}_j \epsilon \underline{D}_k} d_j^Q \end{aligned}$$

Dieser theoretische Vorteil ist aber (im Gegensatz zu Datenbanksystemen) von geringer praktischer Bedeutung; da ein Benutzer in der Regel nicht genau weiß, wie die zu seiner Frage relevanten Dokumente aussehen, kann er auch die Anfrage nicht entsprechend der hier skizzierten Vorgehensweise formulieren.

5.3.2 Nachteile des booleschen Retrieval

In der IR-Forschung ist man sich seit langem darüber einig, daß das boolesche Modell ziemlich ungeeignet für die Anwendung im IR ist [Verhoeff et al. 61]. In [Salton et al. 83] werden folgende Nachteile für boolesches Retrieval genannt:

1. Die Größe der Antwortmenge ist schwierig zu kontrollieren.
2. Es erfolgt keine Ordung der Antwortmenge nach mehr oder weniger relevanten Dokumenten.
3. Es gibt keine Möglichkeit zur Gewichtung von Fragetermen oder zur Berücksichtigung von gewichteter Indexierung.
4. Die Trennung in gefundene und nicht gefundene Dokumente ist oftmals zu streng:
   Zu $q = t_1 \wedge t_2 \wedge t_3$ werden Dokumente mit zwei gefundenen Termen genauso zurückgewiesen wie solche mit 0 gefundenen Termen.
   Analog erfolgt für $q = t_1 \vee t_2 \vee t_3$ keine Unterteilung der gefundenen Dokumente
5. Die Erstellung der Frageformulierung ist sehr umständlich und überfordert daher gelegentliche Benutzer.

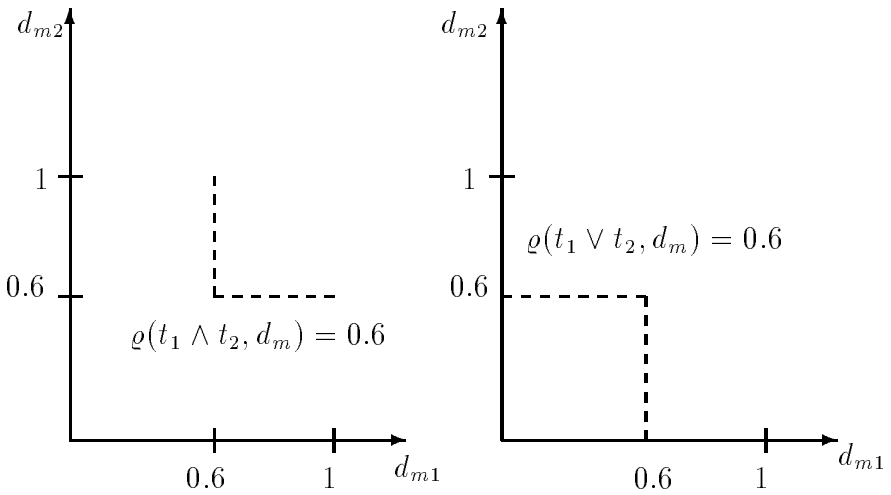


Abbildung 5.3: Punkte mit gleichem Retrievalgewicht beim Fuzzy-Retrieval

6. Die Retrievalqualität von booleschem Retrieval ist wesentlich schlechter als die von anderen Retrievalmodellen (s. nächster Abschnitt).

5.4 Fuzzy-Retrieval

Als ein Ansatz, um einige der Nachteile von booleschem Retrieval zu überwinden, wurde basierend auf der Theorie der Fuzzy-Logik [Zadeh 65] Fuzzy-Retrieval vorgeschlagen. Im Unterschied zum booleschen Modell werden hier bei den Dokumenten-Beschreibungen nun auch gewichtete Indexierungen zugelassen, d.h. $d_{m_i} \epsilon [0,1]$. Frage-Beschreibungen und Retrievalfunktion sind wie beim booleschen Retrieval definiert. Durch die gewichtete Indexierung liefert die Retrievalfunktion jetzt Werte $\varrho(q_k^D, \vec{d}_m) \epsilon [0,1]$. Damit ergibt sich im Gegensatz zum booleschen Modell nun eine Rangordnung der Antwortdokumente, und die diesbezüglichen Nachteile des booleschen Retrieval entfallen. Theoretische Überlegungen wie auch experimentelle Untersuchungen zeigen aber, daß die Definition der Retrievalfunktion ungünstig ist. Wir illustrieren dies zunächst an einem Beispiel:

$$
\begin{aligned}
T &= \{t_1, t_2\} \\
q &= t_1 \wedge t_2 \\
\vec{d}_1 = (0.4, 0.4) \quad &, \quad \vec{d}_2 = (0.39, 0.99) \\
\varrho(q, \vec{d}_1) = 0.4 \quad &, \quad \varrho(q, \vec{d}_2) = 0.39
\end{aligned}
$$

Obwohl hier d_2 bezüglich t_2 ein deutlich höheres Indexierungsgewicht als d_1 hat, gibt das um 0.01 niedrigere Gewicht bzgl. t_1 den Ausschlag für das insgesamt höhere Retrievalgewicht von d_1. Der Grund hierfür ist die Verwendung der Minimum-Funktion

Kollektion	MEDLARS	ISI	INSPEC	CACM
#Dok.	1033	1460	12684	3204
#Fragen	30	35	77	52
Bool.	0.2065	0.1118	0.1159	0.1789
Fuzzy	0.2368	0.1000	0.1314	0.1551
Vektor	0.5473	0.1569	0.2325	0.3027

Tabelle 5.1: Experimenteller Vergleich von Booleschem Retrieval, Fuzzy-Retrieval und Vektorraummodel

bei der konjunktiven Verknüpfung. In der Abb. 5.4 ist jeweils für Konjunktion und Disjunktion die Menge aller Paare von Gewichten (d_{m_1}, d_{m_2}) markiert, für die sich ein Retrievalgewicht von 0.6 ergibt. Offensichtlich wäre es wünschenswert, wenn man zumindest eine teilweise Kompensation der Gewichte für die verschiedenen Terme aus der Anfrage zulassen würde. Ein solches Modell ist unter dem Namen "Extended Boolean Retrieval" in [Salton et al. 83] vorgestellt worden.

In der gleichen Veröffentlichung werden auch experimentelle Ergebnisse zum Vergleich von booleschen und Fuzzy-Retrieval mit dem Vektorraummodell präsentiert. Tabelle 5.1 zeigt diese Ergebnisse in Form mittlerer Precision-Werte für die Recall-Punkte 0.25, 0.5 und 0.75. (Das teilweise schlechtere Abschneiden von Fuzzy- gegenüber booleschem Retrieval ist dabei wohl auf die verwendete Evaluierungsmethode zurückzuführen, die für mehrere Dokumente im gleichen Rang ungeeignet ist.)

5.4.1 Beurteilung des Fuzzy-Retrieval

Zusammengefaßt bietet Fuzzy-Retrieval folgende Vor- und Nachteile:

+ Durch Generalisierung des booleschen Retrieval für gewichtete Indexierung ergibt sich eine Rangordnung der Dokumente.

– Der Ansatz erlaubt zunächst keine Fragetermgewichtung. Es wurden zwar einige Vorschläge hierzu gemacht (siehe den Überblick in [Bookstein 85]), die aber allesamt wenig überzeugen; zudem wurde keiner dieser Ansätze evaluiert. Den besten Vorschlag zur Behandlung dieser Problematik stellt das oben erwähnte "Extended Boolean Retrieval" dar.

– Die Retrievalqualität ist immer noch schlecht im Vergleich z.B. zum Vektorraummodell.

– Da die Frageformulierungen die gleichen wie beim booleschen Retrieval sind, bleibt der Nachteil der umständlichen Formulierung bestehen.

5.5 Das Vektorraummodell

Das Vektorraummodell (VRM) ist wahrscheinlich das bekannteste Modell aus der IR-Forschung. Es wurde ursprünglich im Rahmen der Arbeiten am SMART-Projekt entwickelt [Salton 71]. SMART ist ein experimentelles Retrievalsystem, das von Gerard Salton und seinen Mitarbeitern seit 1961 zunächst in Harvard und später in Cornell entwickelt wurde. In den 80er Jahren wurde das Modell nochmals von Wong und Raghavan überarbeitet [Raghavan & Wong 86].
Im VRM werden Dokumente und Fragen (bzw. deren Beschreibungen) als Punkte in einem Vektorraum aufgefaßt, der durch die Terme der Datenbasis aufgespannt wird. Beim Retrieval wird dann nach solchen Dokumenten gesucht, deren Vektoren ähnlich (im Sinne einer vorgegebenen Metrik) zum Fragevektor sind. Durch diese geometrische Interpretation ergibt sich ein sehr anschauliches Modell.
Der zugrundeliegende Vektorraum wird als orthonormal angenommen, d.h.

- alle Term-Vektoren sind orthogonal (und damit auch linear unabhängig), und
- alle Term-Vektoren sind normiert.

Diese Annahmen stellen natürlich eine starke Vereinfachung gegenüber den realen Verhältnissen dar. (In [Wong et al. 87] wird alternativ hierzu versucht, explizit einen solchen orthonormalen Vektorraum zu konstruieren, dessen Dimensionalität deutlich niedriger als $|T|$ ist.)
Die im VRM zugrundegelegte Dokument-Beschreibung ist ähnlich der des Fuzzy-Retrieval eine gewichtete Indexierung; allerdings sind hier neben Gewichten größer als 1 prinzipiell auch negative Gewichte zulässig (obwohl negative Gewichte in SMART nie verwendet werden):

$$d_m^D = \vec{d}_m \quad \text{mit} \quad d_{m_i} \epsilon \mathbb{R} \quad \text{für } i = 1, \ldots, n \tag{5.2}$$

Die Frage-Beschreibungen haben die gleiche Struktur wie die Dokument-Beschreibungen:

$$q_k^Q = \vec{q}_k \quad \text{mit} \quad q_{k_i} \epsilon \mathbb{R} \quad \text{für } i = 1, \ldots, n \tag{5.3}$$

Als Retrievalfunktion werden verschiedene Vektor-Ähnlichkeitsmaße (z.B. das Cosinus-Maß) angewendet. Meistens wird mit dem Skalarprodukt gearbeitet:

$$\varrho(\vec{q}_k, \vec{d}_m) = \vec{q}_k \cdot \vec{d}_m \tag{5.4}$$

Das folgende Beispiel illustriert die Anwendung des VRM:
Beispiel-Frage: „side effects of drugs on memory and cognitive abilities, not aging“
Entsprechend den Retrievalgewichten werden die Dokumente in der Reihenfolge d_3, d_1, d_4, d_2 ausgegeben.

t_i	q_{k_i}	d_{1_i}	d_{2_i}	d_{3_i}	d_{4_i}
side effect	2	1	0.5	1	1
drugs	2	1	1	1	1
memory	1	1		1	
cognitive ability	1		1	1	0.5
¬ aging	-2		1		
Retrievalgewicht		5	2	6	4.5

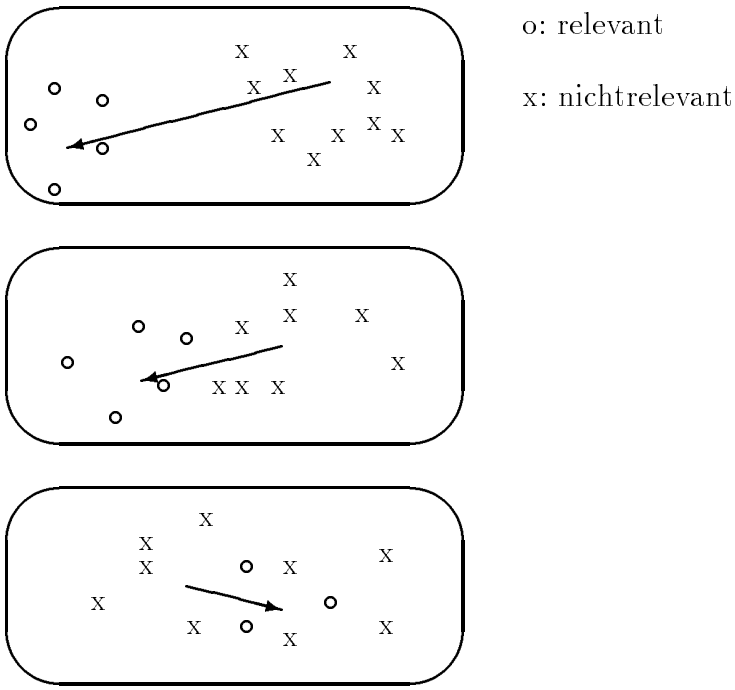


Abbildung 5.4: Trennung von relevanten und nichtrelevanten Dokumenten im VRM

5.5.1 Relevance Feedback

Ein wesentlicher Vorteil des VRM insbesondere auch gegenüber Fuzzy-Retrieval ist die Möglichkeit, Relevance-Feedback-Daten zur Verbesserung der Retrievalqualität auszunutzen. Dabei wird versucht, Angaben über die Relevanz bzw. Nicht-Relevanz einiger Dokumente zur Modifikation des ursprünglichen Fragevektors zu verwenden. Genauer gesagt, werden die ursprünglichen Fragetermgewichte verändert, wodurch sich ein anderer Fragevektor ergibt. Abb. 5.4 illustriert verschiedene mögliche Verteilungen von relevanten und nichtrelevanten Dokumenten im Vektorraum. Außerdem ist jeweils der Vektor eingezeichnet, der vom Zentroiden der nichtrelevanten Dokumente zum Zentroiden der relevanten Dokumente führt. Dieser Vektor eignet sich offensichtlich als Fragevektor, um relevante und nichtrelevante Dokumente möglichst

gut zu trennen. Nimmt man nämlich das Skalarprodukt als Retrievalfunktion an, dann werden die Dokumente auf eine Gerade entlang des Fragevektors projiziert, wobei der Vektor die Richtung höherer Retrievalgewichte anzeigt.
Der Verbindungsvektor $\vec{z}_k$ der beiden Zentroiden zu einer Frage q_k läßt sich wie folgt berechnen: Sei D_k^R die Menge der relevanten Dokumente zu q_k und D_k^N die Menge der nichtrelevanten Dokumente zu q_k. Dann ergibt sich $\vec{z}_k$ zu:

$$\vec{z}_k = \frac{1}{|D_k^R|} \sum_{d_j \epsilon D_k^R} \vec{d}_j - \frac{1}{|D_k^N|} \sum_{d_j \epsilon D_k^N} \vec{d}_j \tag{5.5}$$

Man kann auch theoretisch zeigen, daß dieser Verbindungsvektor relevante und nichtrelevante Dokumente optimal trennt, wenn man als Optimierungskriterium die Bedingung

$$\sum_{(d_i,d_j)\epsilon D^R \times D^N} \vec{q}^T \vec{d}_i - \vec{q}^T \vec{d}_j \stackrel{!}{=} \max \tag{5.6}$$

wählt, d.h., die Differenz der Retrievalgewichte von relevanten und nichtrelevanten Dokumenten soll maximal werden (näheres in der Übung).
In Abb. 5.4 wird bereits deutlich, daß nicht immer (in der Praxis fast nie) eine klare Trennung zwischen den beiden Dokumentklassen möglich ist. Tatsächlich wird man aber nie den Fragevektor allein aus den Relevance-Feedback-Daten ohne Berücksichtigung des ursprünglichen Vektors bilden, da es ja noch weitere Dokumente gibt, über die noch keine Relevanzinformation verfügbar ist, weil das System diese Benutzer noch nicht vorgelegt hat. Gerade diese Dokumente sollen aber möglichst gut in relevante und nichtrelevante aufgeteilt werden — das ist ja die eigentliche Aufgabe beim Retrieval. Daher geht es darum, den ursprünglichen Vektor mit Hilfe der Relevance-Feedback-Daten zu verbessern. Prinzipiell ergibt sich also folgende Vorgehensweise:

1. Retrieval mit dem Fragevektor $\vec{q}_k$ vom Benutzer.
2. Relevanzbeurteilung der obersten Dokumente der Rangordnung.
3. Berechnung eines verbesserten Fragevektors $\vec{q}_k{}'$ aufgrund der Feedback-Daten.
4. Retrieval mit dem verbesserten Vektor.
5. Evtl. Wiederholung der Schritte 2-4.

Als Iterationsvorschrift zur Berechnung eines verbesserten Fragevektors $\vec{q}_k{}'$ wird in [Rocchio 66] folgende Kombination aus ursprünglichem Vektor $\vec{q}_k$ und den Zentroiden der relevanten und der nichtrelevanten Dokumente folgende Formel vorgeschlagen:

$$\vec{q}_k{}' = \vec{q}_k + \alpha \frac{1}{|D_k^R|} \sum_{d_j \epsilon D_k^R} \vec{d}_j - \beta \frac{1}{|D_k^N|} \sum_{d_j \epsilon D_k^N} \vec{d}_j \tag{5.7}$$

Dabei sind α und β nichtnegative Konstanten, die heuristisch festzulegen sind (z.B. $\alpha = \beta = 0.5$).
Tabelle 5.2 zeigt experimentelle Ergebnisse, die durch Anwendung der Formel 5.7 gewonnen wurden (aus [Salton & Buckley 90]). Hier wurde Feedback-Information von

Kollektion	CACM	CISI	CRAN	INSPEC	MED
ohne RF	0.1459	0.1184	0.1156	0.1368	0.3346
Feedback	0.2552	0.1404	0.2955	0.1821	0.5630
Feedback*	0.2491	0.1623	0.2534	0.1861	0.5279

Tabelle 5.2: Experimentelle Ergebnisse zu Relevance Feedback

den obersten 15 Dokumenten des Retrievallaufs mit dem initialen Fragevektor verwendet. Zur Bewertung wurde die sogenannte "residual collection"-Methode angewendet: dabei bleiben die Dokumente, deren Feedback-Daten benutzt wurden, bei der Bewertung unberücksichtigt. Dadurch ergibt sich ein fairer Vergleich mit der Retrievalfunktion ohne Relevance Feedback. Die Ergebnisse zeigen hier sehr deutliche Verbesserungen durch die Relevance-Feedback-Methode. Die letzte Tabellenzeile (Feedback*) zeigt die Ergebnisse für eine modifizierte Anwendung der obigen Formel, bei der nur die häufigsten Terme zur Frageerweiterung benutzt werden, d.h., bei den Termen, deren Fragetermgewicht ursprünglich 0 war (weil sie in der Fragerepräsentation nicht vorkamen), wird die Formel nicht generell in der beschriebenen Weise angewandt; es werden nur die n häufigsten Terme in der vorgeschriebenen Weise berücksichtigt, die übrigen Terme behalten das Gewicht 0. Es zeigt sich, daß diese Methode bei einigen Kollektionen noch zu besseren Ergebnissen führt, während bei anderen Kollektionen schlechtere Ergebnisse produziert werden.

Auch wenn die Formel 5.7 erwiesenermaßen gute Ergebnisse liefert, so sind die heuristischen Komponenten in diesem Ansatz doch unbefriedigend. Letzten Endes liegt die grundlegende Schwäche des VRM in dem fehlenden Bezug zur Retrievalqualität. Auch die o.g. Optimierungsbedingung 5.6 nimmt nicht auf die Retrievalqualität Bezug, und man kann zeigen, daß es tatsächlich in manchen Fällen bessere Vektoren zur Trennung in relevante und nichtrelevante Dokumente gibt, als sie durch diese Bedingung geliefert werden (näheres siehe Übung).

5.5.2 Indexierung in SMART

Das VRM macht keine Aussagen darüber, wie die Dokumentenbeschreibung zu erstellen ist. Im Rahmen der Arbeiten am SMART-Projekt wurden heuristische Formeln zur Berechnung der Indexierungsgewichte für Dokumente (und Fragen) entwickelt, die sich als besonders leistungsfähig erwiesen haben.

Die der Indexierung zugrundeliegende Dokumenten-Repräsentation ist eine Multi-Menge (Bag) von Terms. Darauf aufbauend werden zunächst folgende Parameter definiert:

Kollektion	CACM	CISI	CRAN	INSPEC	MED
Coord.	0.185	0.103	0.241	0.094	0.413
SMART	0.363	0.219	0.384	0.263	0.562

Tabelle 5.3: Binäre Gewichte vs. SMART-Gewichtung

d_m^T Menge der in d_m vorkommenden Terms
tf_{mi} Vorkommenshäufigkeit (Vkh) von t_i in d_m.
$max\, tf_m$ maximale Vkh tf_{mi} aller Terms $t_i \in d_m^T$.
n_i Anzahl Dokumente, in denen t_i vorkommt.
$|\underline{D}|$ Anzahl Dokumente in der Kollektion.

Eine Komponente der Gewichtung ist die inverse Dokumenthäufigkeit idf_i, die umso höher ist, je seltener ein Term in der Kollektion vorkommt:

$$idf_i = \log \frac{|\underline{D}|}{n_i} \tag{5.8}$$

Die zweite Komponente ist die normalisierte Vorkommenshäufigkeit ntf_i. Hierbei sollen die Terms entsprechend ihrer Vorkommenshäufigkeit im Dokument gewichtet werden. Um den Einfluß der Dokumentlänge auszugleichen, wird diese Häufigkeit normalisiert durch die maximale Vorkommenshäufigkeit eines Terms in dem betreffenden Dokument:

$$ntf_i = 0.5(1 + \frac{tf_{mi}}{\max tf_m}) \tag{5.9}$$

Das sogenannte unnormierte Indexierungsgewicht berechnet sich dann als Produkt der beiden Komponenten:

$$\alpha_{mi} = ntf_i \cdot idf_i \tag{5.10}$$

Abschließend wird der Dokumentvektor noch normiert, so daß $|\vec{d_m}| = 1$. Der Normierungsfaktor w_m berechnet sich dabei zu

$$w_m = \sqrt{\sum_{t_i \in d_m^T} \alpha_{mi}^2}. \tag{5.11}$$

Damit ergibt sich das endgültige normierte Indexierungsgewicht

$$d_{mi} = \frac{\alpha_{mi}}{w_m} \tag{5.12}$$

.

Tabelle 5.3 zeigt einige experimentelle Ergebnisse (aus [Salton & Buckley 88]) zu dieser Art der Gewichtung im Vergleich zu einer rein binären Gewichtung (Coordination Level Match). Dabei wurden die Gewichtungsformeln 5.8–5.12 sowohl zur Dokumentindexierung als auch zur Bestimmung des Fragevektors angewendet.

5.5.3 Beurteilung des VRM

Zusammenfassend ergeben sich folgende Vor- und Nachteile für das VRM:

+ Das VRM ist ein relativ einfaches, anschauliches Modell, das insbesondere wegen der einfachen Art der Frageformulierung auch benutzerfreundlich ist.
+ Das Modell ist unmittelbar auf neue Kollektionen anwendbar; probabilistische Modelle erfordern dagegen teilweise zuerst das Sammeln von Relevance-Feedback -Daten für eine Menge von Fragen, bevor sie sinnvoll eingesetzt werden können.
+ Das Modell liefert in Kombination mit den SMART-Gewichtungsformeln eine sehr gute Retrievalqualität.
– Leider enthält das Modell, so wie es letztendlich angewendet wird, sehr viele heuristische Komponenten; dabei stellt sich insbesondere die Frage, inwieweit diese Heuristiken auch noch beim Übergang auf wesentlich andere Kollektionen (z.B. Volltexte statt Kurzfassungen) gültig bleiben.
– Der heuristische Ansatz zur Berechnung der Indexierungsgewichte hat zur Folge, daß die Dokumentrepräsentation nur schlecht erweitert werden kann. Wenn man z.B. Terms aus dem Titel stärker gewichten möchte als solche, die nur im Abstract vorkommen, dann müssen hierfür erst umfangreiche Experimente durchgeführt werden, um eine geeignete Gewichtungsformel zu finden.
– In dem Modell wird keinerlei Bezug auf die Retrievalqualität genommen; es ist theoretisch nicht zu begründen, warum die zu einer Frage ähnlichen Dokumente auch relevant sein sollen.

5.6 Dokumenten-Clustering

Eine von allen anderen Retrievalmodellen gänzlich unterschiedliche Retrievalmethode ist das Cluster-Retrieval. Während andere Retrievalmodelle stets von einer expliziten Frageformulierung ausgehen, nutzt man beim Cluster-Retrieval hauptsächlich die Ähnlichkeit von Dokumenten, um von einem relevanten Dokument zu weiteren (potentiell) relevanten zu gelangen. Diese Art der Suche wird durch die vorherige Bestimmung von (Dokumenten-)Clustern, also Mengen von ähnlichen Dokumenten, unterstützt.
Ausgangspunkt für diese Art der Suche ist die sogenannte „Cluster-Hypothese“: Man kann nämlich zeigen, daß die Ähnlichkeit der relevanten Dokumente untereinander und der irrelevanten Dokumente untereinander größer ist als die zwischen anderen (zufälligen) Teilmengen der Dokumentenkollektion. Diese Hypothese wurde auch experimentell in [Rijsbergen & Sparck Jones 73] nachgewiesen. Beim Dokumenten-Clustering wird versucht, diese Cluster unabhängig von den Fragen schon beim Aufbau der Kollektion zu berechnen. Dabei geht man prinzipiell wie folgt vor:

1. Festlegung eines Ähnlichkeitsmaßes (z.B. Skalarprodukt oder Cosinus-Maß) .

2. Berechnung der Ähnlichkeitmatrix für alle möglichen Dokumentenpaare aus $|\underline{D}|$.
3. Berechnung der Cluster.
4. Physisch gemeinsame Abspeicherung der Dokumente eines Clusters. (Durch diese Form der Speicherung werden I/O-Zugriffe beim Retrieval gespart.)

Zur Berechnung der Cluster aus der Ähnlichkeitsmatrix gibt es eine ganze Reihe von Cluster-Algorithmen (siehe die einschlägige Literatur). Ein einfacher Algorithmus hierzu sieht etwa so aus:

1. Wahl eines Schwellenwertes für die Ähnlichkeit.
2. Alle Dokumentpaare, deren Ähnlichkeit den Schwellenwert überschreitet, werden durch eine Kante verbunden.
3. Die maximalen Cliquen (zusammenhängende Komponenten) des resultierenden Graphen bilden die gesuchten Cluster.

Dieser Algorithmus hat den Aufwand: $O(n^2)$. Es gibt aber auch iterative Clustering-Methoden, die ohne die vollständige Berechnung der Ähnlichkeitsmatrix auskommen und einen Aufwand von $O(n \log n)$ oder $O(n^2/\log n)$ haben.
Die so berechneten Cluster können auf zwei Arten genutzt werden:

1. Zur Suche ähnlicher Dokumente zu einem bereits bekannten relevanten Dokument.
2. Wenn man noch kein relevantes Dokument kennt, wird zunächst Cluster-Retrieval durchgeführt. Hierbei wird ein anderes Retrievalmodell (üblicherweise VRM) angewendet, um Cluster mit potentiell relevanten Dokumenten zu lokalisieren.

5.6.1 Cluster-Retrieval

Beim Cluster-Retrieval wird bei der Berechnung der Cluster zu jedem Cluster ein Zentroid berechnet. Dieser Zentroid ist ein virtuelles Dokument mit minimalem Abstand zu allen Dokumenten des Clusters. Alle Zentroiden werden gemeinsam abgespeichert, und zwar getrennt von den eigentlichen Clustern. Dadurch erspart man sich I/O-Zugriffe beim Durchsuchen der Zentroiden.
Beim eigentlichen Retrieval geht man dann wie folgt vor:

1. Bestimmung der Zentroiden mit den höchsten Retrievalgewichten.
2. Ranking der Dokumente in den zugehörigen Clustern.

Abbildung 5.5 illustriert diese Vorgehensweise. Die Zentroiden sind als Kreise dargestellt, die Dokumente als Quadrate. Bei Dokumenten und Zentroiden ist das Retrievalgewicht jeweils in Klammern notiert. Nur die Cluster zu den grau markierten Zentroiden werden in das abschließende Ranking einbezogen.
Insgesamt ergibt sich folgende Beurteilung für das Cluster-Retrieval:

\+ Die Abhängigkeiten zwischen Dokumenten werden berücksichtigt. Fast alle anderen IR-Modelle nehmen dagegen die Dokumente als voneinander unabhängig an, was natürlich in der Realität nicht stimmt.

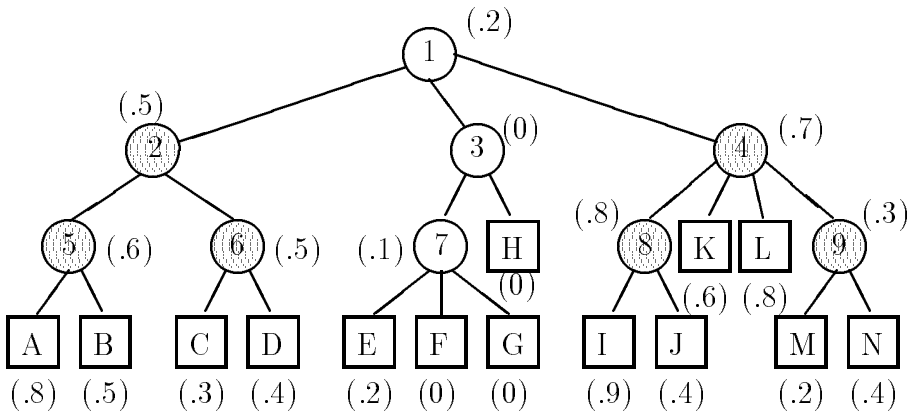


Abbildung 5.5: Beispiel zum Cluster-Retrieval

+ Im Vergleich zu anderen Retrievalverfahren reduziert sich der I/O-Aufwand.
– Soweit experimentelle Ergebnisse zum Cluster-Retrieval vorliegen, zeigen diese eine deutlich schlechtere Retrievalqualität im Vergleich zu anderen Verfahren.

5.6.2 Ähnlichkeitssuche von Dokumenten

Wenn bereits ein relevantes Dokument bekannt ist, dann können die Dokumenten-Cluster zur Ähnlichkeitssuche ausgenutzt werden. Dies ist bei realen Benutzungen von IR-Systemen häufig der Fall. Statt nun eine entsprechende Anfrage zu formulieren, kann man dann eine Ähnlichkeitssuche durchführen. (Heutige kommerzielle IR-Systeme bieten diese Funktion leider nicht.) Auch wenn keine Dokumenten-Cluster vorliegen, kann man natürlich nach ähnlichen Dokumenten suchen: Beim Vektorraummodell faßt man dann einfach den Dokumentvektor des bekannten Dokumentes als Fragevektor auf und führt damit Retrieval durch. Vergleicht man nun den Berechnungsaufwand für diese beiden Vorgehensweisen, so zeigt sich, daß eine Vorprozessierung der Cluster nicht lohnt [Willett 88]. Bezüglich der resultierenden Retrievalqualität gibt es zwar kaum aussagekräftige Resultate; man ist aber allgemein der Auffassung, daß die Ähnlichkeitssuche sinnvoll ist als Ergänzung zu den anderen Retrievalmodellen. Es zeigt sich nämlich, daß hiermit andere relevante Dokumente gefunden werden.

5.6.3 Cluster-Browsing

Wenn Clustering für normales Retrieval gemäß dem oben gesagten wenig Vorteile bietet, so ermöglicht es aber eine alternative Suchstrategie: Während sonst beim Retrieval zunächst der Benutzer eine möglichst konkrete Anfrage formulieren muß, kann man bei einem geclusterten Dokumentenbestand auch einfach browsen. Eine

einfache Möglichkeit hierzu bietet hierarchisches Clustern (wie in Abb. 5.5), wo der Benutzer ausgehend von der Wurzel Pfade zu den einzelnen Dokumenten verfolgt. Für jedes Cluster müssen dann geeignete Repräsentanten angezeigt werden (im Gegensatz zum Cluster-Retrieval wären die berechneten Zentroiden für den Benutzer wenig hilfreich), z.B. indem man das dem Zentroiden ähnlichste Dokument des Clusters nimmt. Dem Benutzer werden dann jeweils die Repräsentanten der Teilcluster des vorher ausgewählten Clusters angezeigt.

5.6.4 Scatter/Gather-Browsing

Eine andere mögliche Anwendung des Clustering wird in [Cutting et al. 92] vorgeschlagen. Die wesentliche Idee hierbei ist, daß Cluster nicht statisch berechnet werden, sondern dynamisch während der interaktiven Suche. Jeder Suchschritt besteht dabei aus zwei Phasen, einer Scatter-Phase und einer Gather-Phase. In der Scatter-Phase wird die Dokumentmenge in eine vorgegebene Anzahl von Clustern zerlegt und dem Benutzer Repräsentanten für die resultierenden Cluster angezeigt. In der Gather-Phase wählt der Benutzer einen oder mehrere ihn interessierende Cluster aus, die dann zusammengemischt die Ausgangsmenge für den nächsten Suchschritt bilden. Abbildung 5.6 zeigt eine Folge von drei Suchschritten. Scatter/Gather-Browsing kann somit als Kombination von Browsing in (statischen) Clustern und Relevance Feedback aufgefaßt werden.
Da Scatter/Gather-Browsing relativ effizient implementiert werden kann, ist das Hauptproblem die Auswahl geeigneter Repräsentanten. Leider gehen die Autoren mit keinem Wort darauf ein, wie etwa die in Abbildung 5.6 gezeigten Bezeichnungen bestimmt wurden. Somit bleiben erhebliche Zweifel, ob diese Methode für Textretrieval praktisch einsetzbar ist. Denkbar ist allerdings die Verwendung für Bildretrieval, wo ein Benutzer viel schneller ein einzelnes Dokument beurteilen kann.

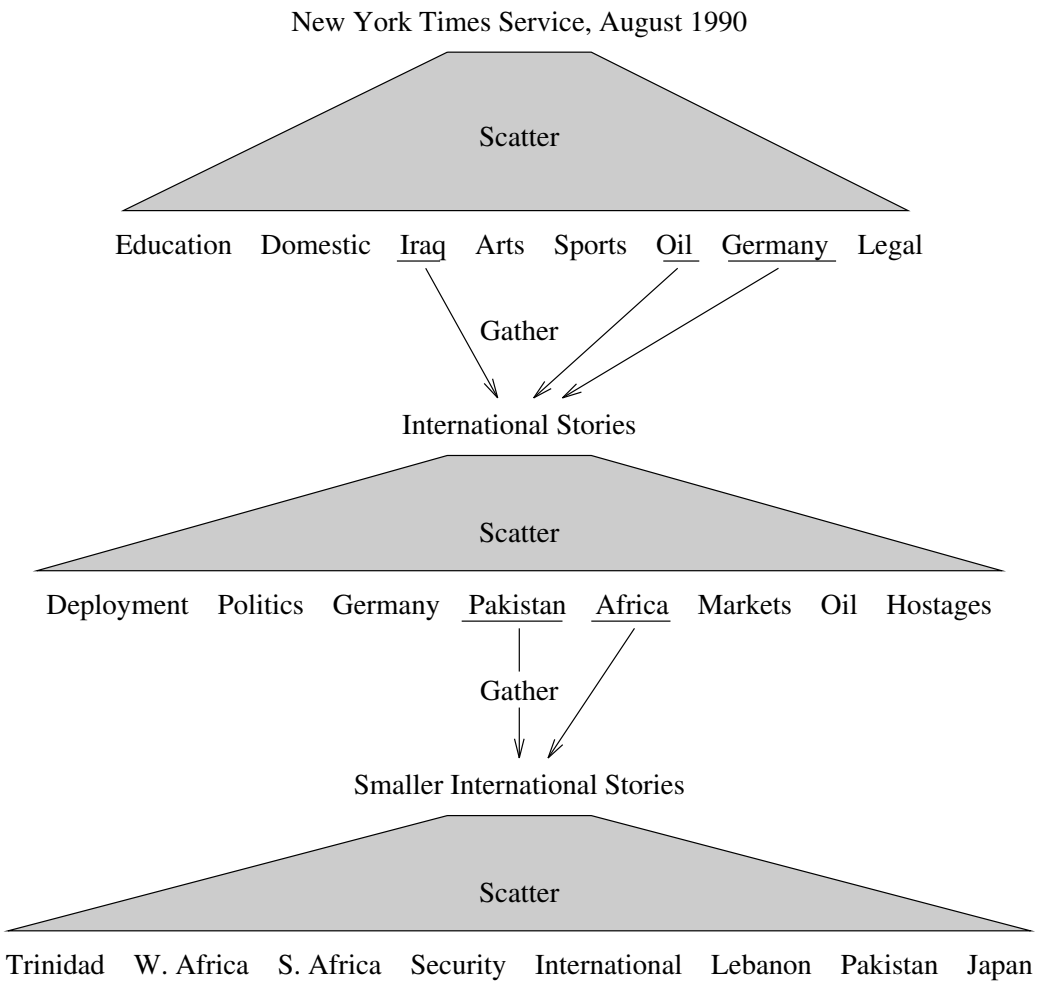


Abbildung 5.6: Beispiel zu Scatter/Gather

Chapter 6

Probabilistic Models in Information Retrieval

6.1 Introduction

A major difference between information retrieval (IR) systems and other kinds of information systems is the intrinsic uncertainty of IR. Whereas for database systems, an information need can always (at least for standard applications) be mapped precisely onto a query formulation, and there is a precise definition of which elements of the database constitute the answer, the situation is much more difficult in IR; here neither a query formulation can be assumed to represent uniquely an information need, nor is there a clear procedure that decides whether a DB object is an answer or not. (Boolean IR systems are not an exception from this statement; they only shift all problems associated with uncertainty to the user.) As the most successful approach for coping with uncertainty in IR, probabilistic models have been developed.
According to the definition of the desired set of answers to a query in an IR system, two major approaches in probabilistic IR can be distinguished: The classical approach is based on the concept of relevance, that is, a user assigns relevance judgements to documents w.r.t. his query, and the task of the IR system is to yield an approximation of the set of relevant documents. The new approach formulated by van Rijsbergen overcomes this subjective definition of an answer in an IR system by generalizing the proof-theoretic model of database systems towards uncertain inference.
In this chapter, an introduction into past and current research in probabilistic IR is given. The major goal here is to present important concepts of this field of research, while no attempt is made to give a complete survey over work in this area. In the following, the classical approach in probabilistic IR is presented in sections 6.2 and 6.3, while section 6.4 describes the new direction. In section 6.5, some general problems of both approaches are discussed. An outlook to future research areas finishes the chapter.

6.2 Basic concepts of relevance models

6.2.1 The binary independence retrieval model

In order to introduce some basic concepts of the classical approach to probabilistic IR, we first present a fairly simple model, the so-called binary independence retrieval (BIR) model. This model will be introduced more informally, whereas the precise assumptions underlying this model will be developed throughout the following sections. In the BIR model, as in most other probabilistic IR models, we seek to estimate the probability that a specific document d_m will be judged relevant w.r.t. a specific query q_k. In order to estimate this probability (denoted as $P(R|q_k, d_m)$ in the following), we regard the distribution of terms within the documents of the collection. (In general, a term is any non-trivial word reduced to its word stem, but see also section 6.5.3 for other kinds of terms). The basic assumption is that terms are distributed differently within relevant and non-relevant documents. This assumption known as the "cluster hypothesis" has been verified experimentally already in [Rijsbergen & Sparck Jones 73]. Let $T = \{t_1, \ldots, t_n\}$ denote the set of terms in the collection. Then we can represent the set of terms d_m^T occurring in document d_m as a binary vector $\vec{x} = (x_1, \ldots, x_n)$ with $x_i = 1$, if $t_i \epsilon d_m^T$ and $x_i = 0$ otherwise.
Now we distinguish only between documents containing different sets of terms, so instead of estimating $P(R|q_k, d_m)$ for a specific document d_m, we actually estimate the probability $P(R|q_k, \vec{x})$, where different documents containing the same set of terms will yield the same estimate of probability of relevance. In addition, the BIR model assumes a query q_k to be just a set of terms $q_k^T \subset T$. In section 6.4.2, we will also discuss the case of other forms of queries.
In order to derive a formula for this probability, we will apply two kinds of transformations that are frequently used for the derivation of probabilistic IR models:

1. application of Bayes' theorem (in the form $P(a|b) = P(b|a) \cdot P(a)/P(b)$),
2. usage of odds instead of probabilities, where $O(y) = P(y)/P(\bar{y}) = P(y)/[1 - P(y)]$.

This way, we can compute the odds of a document represented by a binary vector $\vec{x}$ being relevant to a query q_k as

$$O(R|q_k, \vec{x}) = \frac{P(R|q_k, \vec{x})}{P(\bar{R}|q_k, \vec{x})} = \frac{P(R|q_k)}{P(\bar{R}|q_k)} \cdot \frac{P(\vec{x}|R, q_k)}{P(\vec{x}|\bar{R}, q_k)} \tag{6.1}$$

Now additional independence assumptions are needed in order to arrive at a formula that is applicable for retrieval of documents. As it has been pointed out in a recent paper by Cooper [Cooper 91], the assumption underlying the BIR is in fact not a

set of independence assumptions (from which the name of the model is derived), but rather the assumption of linked dependence of the form

$$\frac{P(\vec{x}|R,q_k)}{P(\vec{x}|\bar{R},q_k)} = \prod_{i=1}^{n} \frac{P(x_i|R,q_k)}{P(x_i|\bar{R},q_k)} \tag{6.2}$$

This assumption says that the ratio between the probabilities of $\vec{x}$ occurring in the relevant and the nonrelevant documents is equal to the product of the corresponding ratios of the single terms. Of course, the linked dependence assumption does not hold in reality. However, it should be regarded as a first-order approximation. In section 6.3.1.3, we will discuss better approximations.
With assumption (6.2), we can transform (6.1) into

$$O(R|q_k,\vec{x}) \quad = \quad O(R|q_k) \prod_{i=1}^{n} \frac{P(x_i|R,q_k)}{P(x_i|\bar{R},q_k)}$$

The product of this equation can be split according to the occurrence of terms in the current document:

$$O(R|q_k,\vec{x}) = O(R|q_k) \prod_{x_i=1} \frac{P(x_i{=}1|R,q_k)}{P(x_i{=}1|\bar{R},q_k)} \cdot \prod_{x_i=0} \frac{P(x_i{=}0|R,q_k)}{P(x_i{=}0|\bar{R},q_k)}.$$

Now let $p_{ik} = P(x_i{=}1|R,q_k)$ and $q_{ik} = P(x_i{=}1|\bar{R},q_k)$. In addition, we assume that $p_{ik} = q_{ik}$ for all terms not occuring in the set q_k^T of query terms. This assumption is also subject to change, as discussed in section 6.5.2. With these notations and simplifications, we arrive at the formula

$$O(R|q_k,\vec{x}) \quad = \quad O(R|q_k) \prod_{t_i \epsilon d_m^T \cap q_k^T} \frac{p_{ik}}{q_{ik}} \prod_{t_i \epsilon q_k^T \setminus d_m^T} \frac{1-p_{ik}}{1-q_{ik}} \tag{6.3}$$

$$\quad = \quad O(R|q_k) \prod_{t_i \epsilon d_m^T \cap q_k^T} \frac{p_{ik}(1-q_{ik})}{q_{ik}(1-p_{ik})} \prod_{t_i \epsilon q_k^T} \frac{1-p_{ik}}{1-q_{ik}} \tag{6.4}$$

In the application of this formula, one is mostly interested only in a ranking of the documents with respect to a query, and not in the actual value of the probability (or odds) of relevance. From this point of view, since the second product of eqn (6.4) as well as the value of $O(R|q_k)$ are constant for a specific query, we only have to consider the value of the first product for a ranking of the documents. If we take the logarithm of this product, the retrieval status value (RSV) of document d_m for query q_k is computed by the sum

$$\sum_{t_i \epsilon d_m^T \cap q_k^T} c_{ik} \qquad \text{with} \qquad c_{ik} = \log \frac{p_{ik}(1-q_{ik})}{q_{ik}(1-p_{ik})}.$$

Then documents are ranked according to descending RSVs.
In order to apply the BIR model, we have to estimate the parameters p_{ik} and q_{ik} for the terms $t_i \epsilon q_k^T$. This can be done by means of relevance feedback. For that, let us assume that the IR system has already retrieved some documents for query q_k (in section 6.5.1, we show how the parameters of the BIR model can be estimated without relevance information). Now the user is asked to give relevance judgements for these documents. From this relevance feedback data, we can estimate the parameters of the BIR model as follows: Let f denote the number of documents presented to the user, of which r have been judged relevant. For a term t_i, f_i is the number among the f documents in which t_i occurs, and r_i is the number of relevant documents containing t_i. Then we can use the estimates $p_{ik} \approx r_i/r$ and $q_{ik} \approx (f_i - r_i)/(f - r)$. (Better estimation methods are discussed in section 6.5.1).
We illustrate this model by giving an example. Assume a query q containing two terms, that is $q^T = \{t_1, t_2\}$. Table 6.1 gives the relevance judgements from 20 documents together with the distribution of the terms within these documents.

d_i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
x_1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
x_2	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0
$r(q, d_i)$	R	R	R	R	$\bar{R}$	R	R	R	R	$\bar{R}$	$\bar{R}$	R	R	R	$\bar{R}$	$\bar{R}$	$\bar{R}$	R	$\bar{R}$	$\bar{R}$

Table 6.1: Example for the BIR model

	$P(R\|q, \vec{x})$	
$\vec{x}$	BIR	actual
(1,1)	0.76	0.8
(1,0)	0.69	0.67
(0,1)	0.48	0.5
(0,0)	0.4	0.33

Table 6.2: Estimates for the probability of relevance for our example

For the parameters of the BIR model, we get $p_1 = 8/12 = 2/3$, $q_1 = 3/8$, $p_2 = 7/12$ and $q_2 = 4/8$. So we get the query term weights $c_1 = \ln 10/3 \approx 1.20$ and $c_2 = \ln 7/5 \approx 0.33$. Based on these weights, documents are ranked according to their corresponding binary vector $\vec{x}$ in the order $(1,1)$ – $(1,0)$ – $(0,1)$ – $(0,0)$. Obviously this ranking is correct for our example.
In addition, we can also compute the estimates for the probability of relevance according to eqn (6.3). With $O(R|q) = 12/8$, we get the estimates shown in table 6.2 for the different $\vec{x}$ vectors , where they are compared with the actual values. Here,

the estimates computed by the two methods are different. This difference is due to the linked dependence assumption underlying the BIR model.
We have described this model in detail because it illustrates a number of concepts and problems in probabilistic IR. In the following, we will first describe the major concepts.

6.2.2 A conceptual model for IR

In our example, it may be argued that the approach chosen in the BIR model for representing documents may be rather crude, and that a more detailed representation of documents may be desirable, especially since documents in an IR system may be rather complex. The relationship between documents and their representations (and similarly for queries) can be illustrated best by regarding the conceptual IR model depicted in figure 6.1.

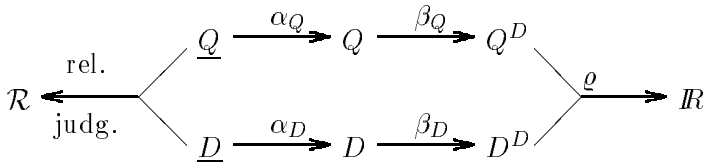


Figure 6.1: Conceptual model

Here $\underline{d_m}$ and $\underline{q_k}$ denote the original document and query, respectively. In our terminology, a query is unique (i.e. a specific information need of a specific user), so two queries from different users (or issued from the same user at different times) can never be identical. This concept of a query has been introduced first in [Robertson et al. 82], where it was termed a "use". Between a document and a query, there exists a relevance relationship as specified by the user. Let $\mathcal{R} = \{R, \bar{R}\}$ denote the set of possible relevance judgements, then the relevance relationship can be regarded as a mapping $r : \underline{Q} \times \underline{D} \rightarrow \mathcal{R}$. Since an IR system can only have a limited understanding of documents and queries, it is based on representations of these objects, here denoted as d_m and q_k. These representations are derived from the original documents and queries by application of the mappings α_D and α_Q, respectively. In the case of the BIR model the representation of a document d_m is a set of terms, namely the set d_m^T. For queries, the representation contains in addition to the set of query terms q_k^T also a set of relevance judgements $q_k^J = \{(\underline{d_m}, r(\underline{d_m}, \underline{q_k}))\}$. So, in our conceptual model, the representation of an object comprises the data relating to this object that is actually used in the model. Different IR models may be based on quite different representations: For example, in Boolean systems with free text search, the document representation is a list of strings (words), and the query representation is a Boolean

expression, where the operands may be either single words or adjacency patterns (comprised of words and special adjacency operators).
For the models regarded here, there is an additional level of representation, which we call description. As can be seen from the BIR model, the retrieval function does not relate explicitely to the query representation, it uses the query term weights derived from the relevance judgements instead. We call the arguments of the retrieval function the description of documents and queries. In the BIR model, the document representation and the description are identical. The query description, however, is a set of query terms with the associated query term weights, that is, $q_k^D = \{(t_i, c_{ik})\}$. Representations are mapped onto descriptions by means of the functions β_D and β_Q, respectively. Based on the descriptions, the retrieval function $\varrho(q_k^D, d_m^D)$ computes the retrieval status value, which is a real number in general. This conceptual model can be applied to probabilistic IR models as well as to other models. Especially when comparing the quality of different models, it is important to consider the representations used within these models. With respect to representations, two directions in the development of probabilistic IR models can be distinguished:

1. Optimization of retrieval quality for a fixed representation. For example, there have been a number of attempts to overcome the limitations of the BIR model by revising the linked dependence assumption and considering certain other forms of term dependencies (see section 6.3.1.3. In these approaches, documents are still represented as sets of terms).
2. Development of models for more detailed representations of queries and documents. Since the document representation used within the BIR model is rather poor, it is desirable to derive models that can consider more detailed information about a term in a document, e.g. its within-document frequency, or the output of advanced text analysis methods (e.g. for phrases in addition to words). We will discuss this issue in section 6.5.3.

6.2.3 Parameter learning in IR

We can make another observation with the BIR model: this model makes very poor use of the relevance feedback information given by the user, since this information is only considered in the ranking process for the current query. For a new query, none of this data can be used at all. If we regard probabilistic IR models as (parameter) learning methods, then three different approaches as shown in figure 6.2 can be distinguished. The three axes indicate to what kinds of objects probabilistic parameters may relate to: documents, queries and terms (that is, elements of the representation). In each of the three approaches, we can distinguish a learning phase and an application phase: In the learning phase, we have relevance feedback data for a certain subset $\underline{Q_L} \times \underline{D_L} \times T_L$ of $\underline{Q} \times \underline{D} \times T$ (where T denotes the set of terms in the collection) from which we can derive probabilistic parameters. These parameters can be used

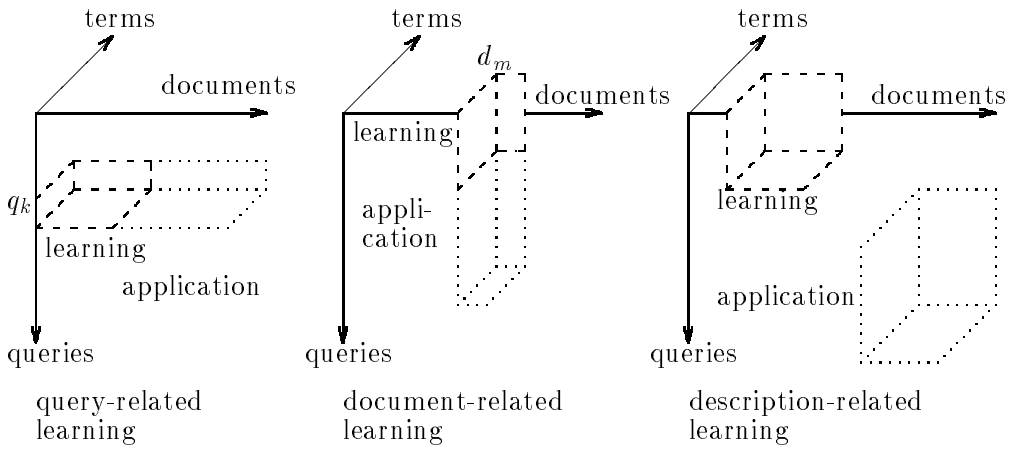


Figure 6.2: Learning approaches in IR

in the application phase for the improvement of the descriptions of documents and queries.

In query-related learning, relevance feedback data is used for weighting of search terms (e.g. in the BIR model) with respect to a single query (representation) q_k. Here we have relevance information from a set of documents $\underline{D_L}$, and we can estimate parameters for the set of terms T_L occurring in these documents. In the application phase, we are restricted to the same query q_k and the set of terms T_L, but we can apply our model to all documents in $\underline{D}$.

Document-related learning is orthogonal to the query-related strategy: probabilistic indexing models (see section 6.3.2) collect relevance feedback data for a specific document d_m from a set of queries $\underline{Q_L}$ with the set of terms T_L occurring in these queries. The parameters derived from this data can be used for the same document and the same set of terms T_L (occurring in queries) only, but for all queries submitted to the system. The major problem with this approach, however, is the fact that there are not enough relevance judgements for a single document in real databases, so it is almost impossible to estimate the parameters of this approach.

The major drawback of this two approaches is their limited application range, since the application phase is either restricted to a single query or to a single document (like in the case of the BII model). In order to overcome these deficiencies, we must introduce abstractions from specific documents, queries and terms. This description-related strategy has been implemented first within the Darmstadt Indexing Approach [Fuhr & Knorz 84] by introducing the concept of relevance descriptions. Similar to pattern recognition methods, a relevance description contains values of features of the objects under consideration (queries, documents and terms). In the learning phase, parameters relating to these features are derived from the learning sample $\underline{Q_L} \times \underline{D_L} \times T_L$. For the application phase, there are no restrictions concerning the

subset $\underline{Q_A} \times \underline{D_A} \times T_A$ of objects to which these parameters can be applied: new queries as well as new documents and new terms can be considered. This strategy is a kind of long-term learning method, since feedback data can be collected from all queries submitted to the IR system, thus increasing the size of the learning sample over time; as a consequence, the probability estimates can be improved. Since this approach is based on descriptions of IR objects instead of the objects itself, we call it description-oriented, in contrast to the model-oriented approaches described before (see also section 6.3.1.3).

6.2.4 Event space

In the presentation of the BIR model, we have not specified the event space to which the probabilities relate to. Now we will define this event space, which is also underlying most probabilistic models.

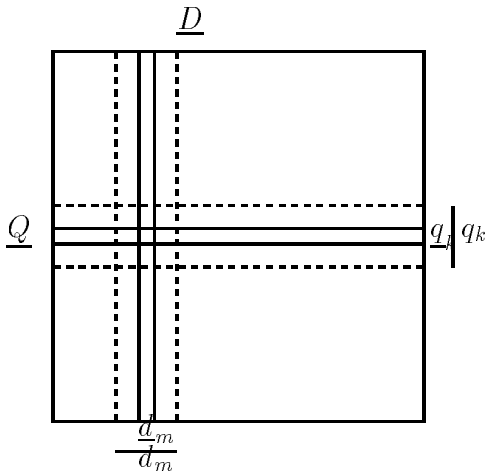


Figure 6.3: Event space of relevance models

The event space is $\underline{Q} \times \underline{D}$. A single element of this event space is a query-document pair $(\underline{q_k}, \underline{d_m})$, where we assume that all these elements are equiprobable. Associated with each element is a relevance judgement $r(\underline{d_m}, \underline{q_k}) \epsilon \mathcal{R}$. We assume that the relevance judgements for different documents w.r.t. the same query are independent of each other. This is a rather strong assumption. Variations of this assumption are discussed in [Robertson 77] and [Stirling 75], but most of these variations lead to models that can hardly be applied in practice. The event space can be illustrated as a matrix as shown in figure 6.3, where a query corresponds to a single row and a document to a column. The relevance judgements can be assumed as the values of the elements of this matrix. Since a retrieval system deals with representations of documents and queries, it treats different queries or documents having identical representations the same. This fact is illustrated here by mapping adjacent rows to a single query representation

q_k and adjacent columns to a single document representation d_m. With this model, the interpretation of the probability of relevance $P(R|q_k, d_m)$ is obvious: the pair (q_k, d_m) corresponds to the set of elements having the same representations (shown as a submatrix here). So $P(R|q_k, d_m)$ is the proportion of elements in this set that have been judged relevant. One might argue that this explanation is oversimplified, since in real collections, there are hardly ever two objects that share the same representation. But we regard collections as samples of possibly infinite sets of documents and queries, where there might be several (up to infinity) objects with the same representation. Especially with regard to the poor representation of retrieval objects that is actually in use (in comparison to the human understanding of documents and queries), it is obvious that a single representation may stand for a number of different objects.

6.2.5 The Probability Ranking Principle

The Probability Ranking Principle (PRP) represents the theoretical justification of probabilistic IR models. It shows how optimum retrieval quality can be achieved. Optimum retrieval is defined w.r.t. representations. In contrast, perfect retrieval relates to the objects itself, saying that all relevant documents should be ranked ahead of any nonrelevant one. But as an IR system is based on representations, perfect retrieval is not a suitable goal. Optimum retrieval has only been defined precisely for probabilistic IR, where the optimality can be proved theoretically. The "Probability Ranking Principle" described in [Robertson 77] says that optimum retrieval is achieved when documents are ranked according to decreasing values of the probability of relevance (with respect to the current query). The decision-theoretic justification of the PRP is as follows (in [Robertson 77], justifications w.r.t. retrieval measures are also given): Let $\bar{C}$ denote the costs for the retrieval of a nonrelevant document, and C the costs for the retrieval of a relevant document. Now the decision-theoretic rule says that document d_m should be retrieved next from the collection if

$$C \cdot P(R|q_k, d_m) + \bar{C} \cdot (1 - P(R|q_k, d_m)) \leq C \cdot P(R|q_k, d_j) + \bar{C} \cdot (1 - P(R|q_k, d_j))$$

for any other document d_j in the collection (that has not been retrieved yet). In other terms: Retrieve that document for which the expected costs of retrieval are a minimum. Because of $C < \bar{C}$, the above condition is equivalent to

$$P(R|q_k, d_m) \geq P(R|q_k, d_j).$$

So we have the rule that documents should be ranked according to their decreasing probability of being relevant. The PRP can be extended to cope with multivalued (ordinal) relevance scales instead of binary ones, as shown in [Bookstein 83b]: Assume that for n relevance values with $R_1 < R_2 < \ldots < R_n$ the corresponding costs for

the retrieval of a document with that retrieval judgement are $C_1, C_2, \ldots, C_n$. Then documents should be ranked according to their expected costs

$$EC(q_k, d_m) = \sum_{l=1}^{n} C_l \cdot P(R_l|q_k, d_m).$$

In contrast to the binary case where only the probability $P(R|q_k, d_m)$ has to be estimated for a query document pair, here $n-1$ estimates $P(R_l|q_k, d_m)$ are needed in order to rank the documents w.r.t. a query. Furthermore, the actual values of the cost factors C_l are required in order to produce a ranking, since they cannot be eliminated as in the binary case. Using multivalued relevance scales instead of binary ones seems to be more appropriate; however, the only experimental results comparing binary vs. multivalued relevance scales published so far did not show any differences in terms of retrieval quality ([Fuhr 89b]). So it might be feasible to offer a multivalued relevance scale for the users of a probabilistic IR system, but this scale can be mapped onto a binary one for the calculations performed by the system.
With multivalued relevance scales, we can also draw a connection to fuzzy retrieval approaches (see [Bookstein 85] for a survey on this subject), where the relevance scale is assumed to be continuous, that is, a relevance judgement now is a real number $r \epsilon [0, 1]$. In this case, the probability distribution $P(R_l|q_k, d_m)$ from above is replaced by a density function $p(r|q_k, d_m)$ as well as the cost factors C_l by a cost function $c(r)$. This way, fuzzy and probabilistic retrieval can be combined. In contrast, pure fuzzy retrieval approaches seem to be inappropriate from the point of view of probabilistic IR, since the intrinsic aspect of uncertainty in IR is ignored in these approaches.

6.3 Some relevance models

6.3.1 A description-oriented approach for retrieval functions

In the description-oriented approach described in the following, retrieval is regarded as a probabilistic classification task. The objects to be classified are request-document pairs $(\underline{q}_k, \underline{d}_m)$, and the classification is the assignment of a value R_l from a relevance scale $\mathcal{R} = (R_1, \ldots, R_n)$.
In this approach, retrieval is performed in two steps:

1. In the **description step**, request-document relationships are mapped onto so-called description vectors $\vec{x} = \vec{x}(q_k, d_m)$.
2. In the **decision step**, a probabilistic classification function is applied in order to compute an estimate of the probabilities $P(R_l|\vec{x}(q_k, d_m))$, $l = 1, \ldots, n$ for a pair $(\underline{q}_k, \underline{d}_m)$. This function is derived from a representative sample of request-document pairs for which relevance judgements are given.

6.3.1.1 Description step

element	description
x_1	# descriptors common to query and document
x_2	log(# descriptors common to query and document)
x_3	highest indexing weight of a common descriptor
x_4	lowest indexing weight of a common descriptor
x_5	# common descriptors with weight ≥ 0.15
x_6	# non-common descriptors with weight ≥ 0.15
x_7	# descriptors in the document with weight ≥ 0.15
x_8	log$\sum$(indexing weights of common descriptors)
x_9	log(# descriptors in the query)

Table 6.3: Example for a description vector $\vec{x}(q_k, d_m)$

In this step, a mapping of request-document pairs onto a description vector $\vec{x} = \vec{x}(q_k, d_m)$ has to be defined. This vector contains values of features of the query, the document and their relationship. The goal is to define features that are correlated with the probability of relevance of the request-document pair. As an example, table 6.3 shows some elements of a description vector (from [Fuhr 89b]). Here, the representation of a document consists of a set of index terms (descriptors) with associated indexing weights, while a query is a set of descriptors.

6.3.1.2 Decision step

For the decision step, a probabilistic classification function yielding estimates of the probabilities $P(R_l|\vec{x}(q_k, d_m))$ have to be derived from a representative sample of request-document pairs with relevance judgements. For this purpose, the relevance judgement $r(\underline{q_k}, \underline{d_m})$ of a request-document pair is represented by a vector $\vec{y}$ with $y_l = 1$, if $r(\underline{q_k}, \underline{d_m}) = R_l$, and $y_l = 0$ otherwise.
Now we seek for a regression function $\vec{e}_{opt}(\vec{x})$ which yields an optimum approximation $\hat{\vec{y}}$ of the class variable $\vec{y}$. As optimizing criterion, we use minimum squared errors here ($E(.)$ denotes the expectation):

$$E(|\vec{y} - \vec{e}_{opt}(\vec{x})|^2) \stackrel{!}{=} \min. \tag{6.5}$$

With this condition, $\vec{e}_{opt}(\vec{x})$ yields an estimate of the probabilities $P(R_l|\vec{x})$, $l = 1, \ldots, n$ (see [Schürmann 77, pp. 163–164]).
Equation 6.5 is the formulation of a general variation problem: Among all possible functions $\vec{e}(\vec{x})$, we seek for the optimum function fulfilling the above criterion. Because of the probabilistic nature of the approach, $\vec{e}_{opt}(\vec{x})$ is the optimum retrieval function (with respect to the chosen vector representation of request-document relationships) that follows the PRP.
However, this variation problem cannot be solved in its general form, so we have to restrict the search to a predefined class of functions. With this restriction, we will only find a sub-optimum solution, but our general variation problem now becomes a parameter optimization task. The resulting functions yield a least squares approximation of $\vec{e}_{opt}$: In [Schürmann 77, pp. 178–180], it is shown that an approximation with respect to the expression $E(|\vec{y} - \hat{\vec{y}}\,|^2) \stackrel{!}{=} \min$ yields the same result as an optimization fulfilling the condition

$$E(|E(\vec{y}\,|\vec{x}\,) - \hat{\vec{y}}\,|^2) \stackrel{!}{=} \min. \tag{6.6}$$

So our restricted optimiziation process yields a least squares approximation of the probabilities $P(\vec{y}|\vec{x}(q_k, d_m))$.

6.3.1.2.1 Least square polynomials

In the least square polynomials (LSP) approach, polynomials with a predefined structure are taken as function classes. For that, a polynomial structure $\vec{v}\,(\vec{x}) = (v_1, \ldots, v_L)$ has to be defined as

$$\vec{v}\,(\vec{x}) = (1, x_1, x_2, \ldots, x_N, x_1^2, x_1 x_2, \ldots)$$

where N is the number of dimensions of $\vec{x}$. The class of polynomials is given by the components $x_i^l \cdot x_j^m \cdot x_k^n \cdot \ldots (i, j, k, \ldots \in [1, N]; l, m, n, \ldots \geq 0)$ which are to be included in the polynomial. In practice, mostly linear and quadratic polynomials are regarded.

The regression function now yields

$$\vec{e}\,(\vec{x}) = A^T \cdot \vec{v}\,(\vec{x}),$$

where $A = (a_{il})$ with i=1, ..., L; l=1, ..., n is the coefficient matrix which has to be estimated. So $P(R_l|\vec{x})$ is approximated by the polynomial

$$e_l(\vec{x}) = a_{1l} + a_{2l} \cdot x_1 + a_{3l} \cdot x_2 + \ldots + a_{N+1,l} \cdot x_N + a_{N+2,l} \cdot x_1^2 + a_{N+3,l} \cdot x_1 \cdot x_2 + \ldots$$

The coefficient matrix is computed by solving the linear equation system [Schürmann 77, pp. 175–176]

$$E(\vec{v} \cdot \vec{v}^{\,T}) \cdot A = E(\vec{v} \cdot \vec{y}^{\,T}). \tag{6.7}$$

As approximation of the expectations, average values from a presentative sample of request-document relationships are used.
The actual computation process is based on the empirical momental matrix $M = (\overline{\vec{v} \cdot \vec{v}^{\,T}}$
$\overline{\vec{v} \cdot \vec{y}^{\,T}})$ which contains both sides of the equation system 6.7. Then the coefficient vector is computed by means of the Gauss-Jordan algorithm [Schürmann 77, pp. 199–227]. For the iterative solution of the equation system, our algorithm always chooses the coefficient which maximizes the reduction of the overall error s^2. Let $M^{(i)} = (m_{lj}^{(i)})$ with l=1, ..., L; j=1, ..., $L + n$ denote the matrix M before the ith elimination step. Then the reduction $d_j^{(i)}$ that will be achieved by choosing component j can be computed from the matrix [Schürmann 77, pp. 209–217] as follows:

$$d_j^{(i)} = \frac{1}{m_{jj}^{(i)2}} \cdot \sum_{l=L+1}^{L+n} m_{jl}^{(i)2}. \tag{6.8}$$

After each iteration, this procedure yields a preliminary solution: With the ith step, we get a polynomial function $\vec{e}^{(i)}(\vec{x})$ with i coefficients which can be regarded as the result of a correspondingly limited optimization process. This property is important in the case of small learning samples, where a limited optimization might yield better retrieval results for other (test) samples than an unlimited one (see below).
We illustrate our approach by giving an example: Assume that we have a binary relevance scale, and that request-document pairs are described by vectors $\vec{x} = (x_1, x_2)$. We define a linear polynomial structure of the form $\vec{v} = (1, x_1, x_2)$. Our learning sample consisting of 8 vectors is shown in Table 6.4. The momental matrix for this sample is

$$M = \frac{1}{8} \cdot \begin{pmatrix} 8 & 5 & 6 & 4 & 4 \\ 5 & 5 & 3 & 3 & 2 \\ 6 & 3 & 6 & 3 & 3 \end{pmatrix}.$$

In order to select the first component to be eliminated, we compute the error reductions by application of eqn 6.8: $d_1^{(1)} = 0.50$, $d_2^{(1)} = 0.52$ and $d_3^{(1)} = 0.50$. So we have

$\vec{x}$	r_k	$\vec{y}$	$P(R_1\|\vec{x})$	$e_1^{(1)}(\vec{x})$	$e_1^{(2)}(\vec{x})$	$e_1^{(3)}(\vec{x})$	$e_1^{(3)'}(\vec{x})$
(1,1)	R_1	(1,0)	0.67	0.6	0.60	0.67	0.69
(1,1)	R_1	(1,0)	0.67	0.6	0.60	0.67	0.69
(1,1)	R_2	(0,1)	0.67	0.6	0.60	0.67	0.69
(1,0)	R_1	(1,0)	0.50	0.6	0.60	0.50	0.46
(1,0)	R_2	(0,1)	0.50	0.6	0.60	0.50	0.46
(0,1)	R_1	(1,0)	0.33	0.0	0.33	0.33	0.31
(0,1)	R_2	(0,1)	0.33	0.0	0.33	0.33	0.31
(0,1)	R_2	(0,1)	0.33	0.0	0.33	0.33	0.31

Table 6.4: Example for the LSP approach

to choose the second component of $\vec{v}$, namely x_1, thus yielding the first polynomials $e_1^{(1)}(\vec{x}) = \frac{3}{5}x_1$ and $e_2^{(1)} = \frac{2}{5}x_1$.
The corresponding values of $e_1^{(1)}$ for the different vectors $\vec{x}$ are depicted in table 6.4. These values are already minimum square error approximations of $P(R_1|\vec{x})$.
Now the matrix $M^{(2)}$ has the value

$$M^{(2)} = \frac{1}{8} \cdot \begin{pmatrix} 3 & 0 & 3 & 1 & 2 \\ 5 & 5 & 3 & 3 & 2 \\ 3 & 0 & 4.2 & 1.2 & 1.8 \end{pmatrix}.$$

For the selection of the next component, we get $d_1^{(2)} = 0.56$ and $d_3^{(2)} = 0.27$. Integrating v_1 into our polynomials we get the functions $e_1^{(2)} = 0.33 + 0.27x_1$ and $e_2^{(2)} = 0.67 - 0.27x_1$.
Finally we have

$$M^{(3)} = \frac{1}{8} \cdot \begin{pmatrix} 3 & 0 & 3 & 1 & 2 \\ 5 & 5 & 3 & 3 & 2 \\ 0 & 0 & 1.2 & 0.2 & -0.2 \end{pmatrix},$$

$e_1^{(3)} = 0.17 + 0.33x_1 + 0.17x_2$ and $e_2^{(3)} = 0.83 - 0.33x_1 - 0.17x_2$.
As can be seen from the comparison of the values of $P(R_1|\vec{x})$, and $e_1^{(3)}(\vec{x})$ in table 6.4, this function yields the correct probabilities. So we have $\vec{e}^{(3)}(\vec{x}) = \vec{e}_{opt}(\vec{x})$ in our example.
But now we add another vector $\vec{x} = (0,0)$ with the relevance judgement $\bar{R}$ to our learning sample, thus yielding the momental matrix

$$M' = \frac{1}{9} \cdot \begin{pmatrix} 9 & 5 & 6 & 4 & 5 \\ 5 & 5 & 3 & 3 & 2 \\ 6 & 3 & 6 & 3 & 3 \end{pmatrix}$$

and $e_1^{(3)'} = 0.08 + 0.38x_1 + 0.23x_2$. For our new vector, we get $e_1^{(3)'}((0,0)) = 0.08$, the other values are listed in table 6.4. Obviously, now $\vec{e}^{(3)'} \neq \vec{e}_{opt}{}'$, that is, our linear

function is only an approximation of $\vec{e}_{opt}{}'$ and thus also yields only approximate values of $P(R_l|\vec{x}\,)$.
There are two important properties of the LSP retrieval functions:

- The basic assumptions of the LSP approach is that we can approximate the expectations by average values. In order to do this, learning samples of sufficient size are required. From previous experience with LSP applications in the IR context [Knorz 83], [Fuhr 88], we can derive a rule of thumb for this size: per component of the polynomial structure $\vec{v}$, there should be about 50–100 elements in the learning sample. With smaller learning samples, there may be parameter estimation problems: as a result, we would get a very effective retrieval function for the learning sample, which would perform significantly worse on other (test) samples. So the definition of $\vec{v}(\vec{x})$ should obey this relationship between vector length and learning sample size. For this reason, it seems to be inappropriate to develop query-specific retrieval functions. Instead, we define request-independent retrieval functions by mapping request-document pairs $(\underline{q_k}, \underline{d_m})$ onto description vectors $\vec{x}(q_k, d_m)$. So we replace query-related learning by description-related learning.
- LSP retrieval functions yield estimates of the probability of relevance $P(R|\vec{x}(q_k, d_m))$. This feature makes the LSP approach very different from other probabilistic models where it is nearly impossible to get these estimates, because there are too many parameters which can be hardly estimated. Therefore other probabilistic models only yield a request-specific ranking following the PRP, but no estimates of $P(R|q_k, d_m)$ can be computed. We think that estimates of the probability of relevance could help a user of an IR system to get some impression of the overall quality of an answer.

6.3.1.3 Model-oriented vs. description-oriented approaches

The formulation of the PRP acts as a goal for any probabilistic IR model. In general, the optimum retrieval quality as specified by the PRP cannot be achieved by a real system. For example, in the BIR model, we would have to know the exact probabilities $P(R|q_k, \vec{x})$ for all binary vectors $\vec{x}$ occurring in the document collection. Except for rather trivial cases, these probabilities can hardly be estimated directly, because the number of different representations is too large in comparison to the amount of feedback data available (so there would not even be any observation for most representations). In order to overcome this difficulty, additional simplifying assumptions are needed. With regard to the nature of these assumptions, two kinds of approaches have been developed:

- Model-oriented approaches (like e.g. the BIR model) are based on certain probabilistic independence assumptions concerning the elements of the representations (e.g. single terms or pairs, triplets of terms). In these approaches,

first probability estimates relating to the representation elements are computed. Then, by applying the independence assumptions, the estimates for the different representations can be derived.

- Description-oriented approaches are similar to feature-based pattern recognition methods. Given the representations of queries and documents, first a set features for query-document pairs is defined, and each pair is mapped onto a feature vector $\vec{x}(q_k, d_m)$. (A specific feature vector could be for example the binary vector $\vec{x} = \vec{x}(d_m)$ as defined in the BIR model; however, since this definition does not consider the query, the resulting retrieval function would be query-specific.) With the help of a learning sample containing query-document pairs with their corresponding relevance judgements, a probabilistic classification function $e(\vec{x})$ that yields estimates of the probability $P(R|\vec{x}(q_k, d_m))$ is developed .

Because of the problem of specifying the feature vector, description-oriented approaches are more heuristical in comparison to model-oriented ones. On the other hand, the assumptions underlying the description-oriented approach do not have to be made as explicit as in the model-oriented case. The most important advantage of description-oriented approaches is their adaptability to rather complex representations, where it is hard to find appropriate independence assumptions. Especially with regard to advanced text analysis methods, this feature seems to be rather important. As a general property of both kinds of approaches, we can see that the additional assumptions are only approximations to reality. For example, we can hardly expect that terms are distributed independently in documents (as suggested by the BIR model). A similar statement holds for the description-oriented approaches. This fact makes the main difference between optimum retrieval quality and the actual performance of a model. The other reason is the problem of parameter estimation. Without going into the details of parameter estimation here (but see section 6.5.1), we can describe the general problem by using the example of the BIR model. The direct estimation of the probabilities $P(R|q_k, \vec{x})$ vs. the computation of this parameter by means of the BIR model are two extreme possibilities where either the probabilities cannot be estimated in a real application or the independence assumptions seem to be too strong. It is possible to develop variants of the BIR model where only pairs or triplets of terms are assumed to be independent of each other (see e.g. [Rijsbergen 79b, Yu et al. 83] for such models and [Pearl 88, chapter 8] for a general survey on probabilistic dependence models). With these models, however, more parameters have to be estimated from less observations for each parameter. For example, in the tree dependence model developed by van Rijsbergen which considers pairwise dependencies [Rijsbergen 77]), the parameters to be estimated for a dependent pair (t_i, t_j) are $P(x_i{=}1, x_j{=}1|R)$, $P(x_i{=}1, x_j{=}0|R)$, $P(x_i{=}0, x_j{=}1|R)$ and $P(x_i{=}0, x_j{=}0|R)$ (plus the corresponding estimates for nonrelevant documents). In contrast, the BIR model only requires the parameters $P(x_i{=}1|R)$ and $P(x_i{=}0|R)$ for the relevant documents, so the tree dependence model splits the learning data required for estimating these

parameters according to the value of x_j. As a consequence, experimental evaluations showed that the gain from improved independence assumptions does not outweigh the loss from increased estimation errors.

6.3.2 The binary independence indexing model

The binary independence indexing (BII) model [Fuhr & Buckley 91] is a variant of the very first probabilistic IR model, namely the indexing model of Maron and Kuhns [Maron & Kuhns 60]. Whereas the BIR model regards a single query w.r.t. a number of documents, the BII model observes one document in relation to a number of queries submitted to the system. In this model, the representation q_k of a query $\underline{q_k}$ is a set of terms $q_k^T \subset T$. As a consequence, the BII model will yield the same ranking for two different queries formulated with the same set of terms. In the following, we will also use a binary vector $\vec{z}_k = (z_{k_1}, \ldots, z_{k_n})$ instead of q_k^T, where $z_{k_i} = 1$, if $t_i \in q_k^T$, and $z_{k_i} = 0$ otherwise. The document representation is not further specified in the BII model, and below we will show that this is a major advantage of this model. In the following, we will assume that there exists a set $d_m^T \subset T$ of terms which are to be given weights w.r.t. the document. For brevity, we will call d_m^T "the set of terms occurring in the document" in the following, although the model also can be applied in situations where the elements of d_m^T are derived from the document text with the help of a thesaurus (see e.g. [Fuhr 89a]). The BII model now seeks for an estimate of the probability $P(R|q_k, d_m) = P(R|\vec{z}_k, d_m)$ that a document with the representation d_m will be judged relevant w.r.t. a query with the representation $q_k = \vec{z}_k$. Applying Bayes' theorem, we first get

$$P(R|\vec{z}_k, d_m) = P(R|d_m) \cdot \frac{P(\vec{z}_k|R, d_m)}{P(\vec{z}_k|d_m)} \tag{6.9}$$

Here $P(R|d_m)$ is the probability that document d_m will be judged relevant to an arbitrary request. $P(\vec{z}_k|R, d_m)$ is the probability that d_m will be relevant to a query with representation $\vec{z}_k$, and $P(\vec{z}_k|d_m)$ is the probability that such a query will be submitted to the system.
Regarding the restricted event space consisting of all documents with the same representation d_m and all queries in the collection, we assume that the distribution of terms in all queries to which a document with representation d_m is relevant is independent:

$$P(\vec{z}_k|R, d_m) = \prod_{i=1}^{n} P(z_{k_i}|R, d_m)$$

With this assumption, eqn 6.9 can be transformed into

$$P(R|\vec{z}_k, d_m) \quad = \quad \frac{P(R|d_m)}{P(\vec{z}_k|d_m)} \cdot \prod_{i=1}^{n} P(z_{k_i}|R, d_m)$$

$$= \frac{P(R|d_m)}{P(\vec{z}_k|d_m)} \cdot \prod_{i=1}^{n} \frac{P(R|z_{k_i}, d_m)}{P(R|d_m)} \cdot P(z_{k_i}|d_m)$$

Since we always regard all documents w.r.t. a query, the probabilities $P(\vec{z}_k|d_m)$ and $P(z_{k_i}|d_m)$ are independent of a specific document. So we get:

$$\begin{aligned} P(R|\vec{z}_k, d_m) &= \frac{\prod_{i=1}^{n} P(z_{k_i})}{P(\vec{z}_k)} \cdot P(R|d_m) \cdot \prod_{i=1}^{n} \frac{P(R|z_{k_i}, d_m)}{P(R|d_m)} \\ &= \frac{\prod_{i=1}^{n} P(z_{k_i})}{P(\vec{z}_k)} \cdot P(R|d_m) \cdot \prod_{z_{k_i}=1} \frac{P(R|z_{k_i}=1, d_m)}{P(R|d_m)} \cdot \\ &\quad \cdot \prod_{z_{k_i}=0} \frac{P(R|z_{k_i}=0, d_m)}{P(R|d_m)} \end{aligned} \tag{6.10}$$

Now we make an additional simplifying assumption that is also used in [Maron & Kuhns 60]: The relevance of a document with representation d_m with respect to a query q_k depends only on the terms from q_k^T, and not on other terms. This assumption means that the last product in formula 6.10 has the value 1 and thus it can be omitted. The value of the first fraction in this formula is a constant c_k for a given query q_k, so there is no need to estimate this parameter for a ranking of documents w.r.t. q_k.
$P(R|z_{k_i} = 1, d_m) = P(R|t_i, d_m)$ is the probabilistic index term weight of t_i w.r.t. d_m, the probability that document d_m will be judged relevant to an arbitrary query, given that it contains t_i. From our model, it follows that d_m^T should contain at least those terms from T for which $P(R|t_i, d_m) \neq P(R|d_m)$. Assuming that $P(R|t_i, d_m) = P(R|d_m)$ for all $t_i \notin d_m^T$, we get the final BII formula

$$P(R|q_k, d_m) = c_k \cdot P(R|d_m) \cdot \prod_{t_i \in q_k^T \cap d_m^T} \frac{P(R|t_i, d_m)}{P(R|d_m)}. \tag{6.11}$$

However, in this form the BII model can hardly be applied, because in general there will not be enough relevance information available for estimating the probabilities $P(R|t_i, d_m)$ for specific term-document pairs. In order to overcome this difficulty, one can assume a document to consist of independent components (e.g. sentences or words) to which the indexing weights relate to, but experimental evaluations showed only moderate retrieval results for this approach ([Kwok 90]).

6.3.3 A description-oriented indexing approach

As a more successful method, the application of the description-oriented approach as outlined in section 6.3.1 has been devised. In this approach, features of terms in documents are regarded instead of the document-term pairs itself. The basic ideas of this approach have been developed within the framework of the Darmstadt Indexing

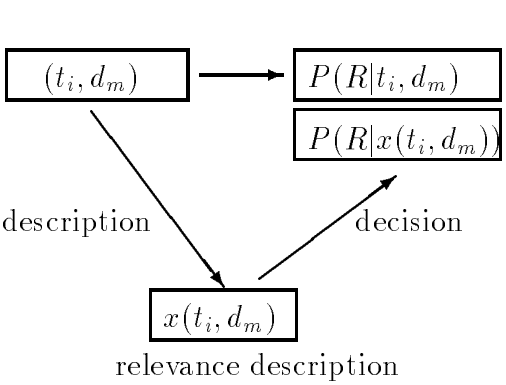


Figure 6.4: Subdivision of the indexing task in a description step and a decision step

Approach (DIA) [Fuhr 89a] [Biebricher et al. 88]. Within the DIA, the indexing task is subdivided in a description step and a decision step (see figure 6.4). In the description step, relevance descriptions for term-document pairs (t_i, d_m) are formed, where a relevance description $x(t_i, d_m)$ contains values of attributes of the term t_i, the document d_m and their relationship. Since this approach makes no additional assumptions about the choice of the attributes and the structure of x, the actual definition of relevance descriptions can be adapted to the specific application context, namely the representation of documents and the amount of learning data available. For example, in the work described in [Fuhr & Buckley 91], the following elements were defined:

x_1 = tf_{mi}, the within-document frequency (wdf) of t_i in d_m
x_2 = the inverse of the maximum wdf of a term in d_m
x_3 = inverse document frequency of t_i in the collection
x_4 = $\log |d_m^T|$ (number of terms in d_m)
x_5 = 1, if t_i occurs in the title of d_m, and 0 otherwise.

In the decision step, a probabilistic index term weight based on this data is assigned. This means that we estimate instead of $P(R|t_i, d_m)$ the probability $P(R|x(t_i, d_m))$. In the former case, we would have to regard a single document d_m with respect to all queries containing t_i in order to estimate $P(R|t_i, d_m)$. But we replace this document-related learning strategy by a description-related one. For that, we regard the set of all query-document pairs in which the same relevance description x occurs. The probabilistic index term weights $P(R|x(t_i, d_m))$ are derived from a learning example $L \subset \underline{Q} \times \underline{D} \times \mathcal{R}$ of query-document pairs for which we have relevance judgements, so $L = \{(\underline{q}_k, \underline{d}_m, r_{km})\}$. By forming relevance descriptions for the terms common to query and document for every query-document pair in L, we get a multi-set (bag) of relevance descriptions with relevance judgements $L^x = [(x(t_i, d_m), r_{km}) | t_i \in q_k^T \cap d_m^T \wedge (\underline{q}_k, \underline{d}_m, r_{km}) \in L]$. From this set with multiple occurrences of elements, the parameters $P(R|x(t_i, d_m))$ could be estimated directly

query	doc.	judg.	term	$\vec{x}$
q_1	d_1	R	t_1	$(1,1)$
			t_2	$(0,1)$
			t_3	$(0,0)$
q_1	d_2	$\bar{R}$	t_1	$(1,1)$
			t_3	$(0,1)$
			t_4	$(0,0)$
q_2	d_1	R	t_3	$(1,1)$
			t_5	$(0,1)$
			t_6	$(1,0)$
q_2	d_3	R	t_5	$(1,1)$
			t_7	$(1,0)$
q_2	d_4	R	t_3	$(1,0)$
			t_6	$(0,1)$
			t_8	$(0,0)$

Table 6.5: Example learning sample

$\vec{x}$	$P(R\|\vec{x})$
$(0,0)$	$1/3$
$(0,1)$	$2/4$
$(1,0)$	$2/3$
$(1,1)$	$3/4$

Table 6.6: Probability estimates for the example from table 6.5

by computing the corresponding relative frequencies.
As a simple example, assume that the relevance description consists of two elements defined as

$$x_1 = \begin{cases} 1, & \text{if } t_i \text{ occurs in the title of } d_m \\ 0, & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 0, & \text{if } t_i \text{ occurs once in } d_m \\ 1, & \text{if } t_i \text{ occurs at least twice in } d_m \end{cases}$$

Table 6.5 shows a small learning sample with queries, documents and relevance judgements. From this data, the probability estimates depicted in table 6.6 can be derived[1].

[1]As discussed in [Fuhr & Buckley 91], there are in fact two different event spaces the probabilities may relate to; either each query-document pair or each relevance description may be equiprobable. However, experimental results showed that this difference can be neglected.

Instead of direct estimation of the parameters $P(R|x(t_i, d_m))$ (as in the example from above), better estimates can be achieved by applying probabilistic classification procedures as developed in pattern recognition or machine learning. Within the DIA, this classification procedure yielding approximations of $P(R|x(t_i, d_m))$ is termed an indexing function $e(x(t_i, d_m))$. Besides the LSP approach described in section 6.3.1.2.1 several probabilistic classification algorithms have been used for this purpose (see e.g. [Fuhr & Buckley 91]).
The major advantage of this indexing approach is its flexibility w.r.t. the representation of documents, which becomes important when advanced text analysis methods are used (e.g. noun phrases in addition to words, see for example [Fuhr 89b]). Experimental results described in [Fuhr & Buckley 91] show that this approach can outperform the best SMART indexing functions (see 5.5.2), provided that relevance feedback data is available.

6.3.4 The 2-Poisson model

On the other hand, one might prefer to have a more explicit model relating to the elements of the representation. One such approach is the 2-Poisson model. This model has been proposed first by Bookstein and Swanson [Bookstein & Swanson 74]. Similar to the indexing model described above, the Bookstein/Swanson model seeks for the decision whether an index term should be assigned to a document or not. So there are two classes of documents with respect to a specific term. Now the number of occurrences tf_{im} of the term t_i within the document d_m is regarded, and it is assumed that the distribution of this feature is different in the two document classes. As a simple probabilistic model, Bookstein and Swanson assumed a Poisson distribution in each of this classes. For a specific document class K_{ij}, let λ_{ij} denote the expectation of the wdf of t_i. So we have $|K_{ij}|$ documents in a class, for which there are altogether $n_{ij} = |K_{ij}| \cdot \lambda_{ij}$ occurrences of terms; these occurrences are spread randomly over the documents of the class. Assuming a Poisson process means that these occurrences are distributed one after the other over the documents, so in each step every document has the same probability (namely $1/|K_{ij}|$) of obtaining the next occurrence of the term. For this Poisson process, the probability that a document obtains altogether l occurrences of t_i is

$$P(tf_{im}{=}l|d_m \epsilon K_{ij}) = \frac{\lambda_{ij}^l}{l!} e^{-\lambda_{ij}}.$$

For a document chosen randomly from the collection, we assume that π_{ij} is the probability that it belongs to class K_{ij}. Then the probability of observing l occurrences within such a document is

$$P(tf_{im}{=}l) = \sum_j \pi_{ij} \frac{\lambda_{ij}^l}{l!} e^{-\lambda_{ij}}.$$

In the 2-Poisson model, there are two document classes K_{i1} and K_{i2} for each term, so $\pi_{i1}+\pi_{i2} = 1$. From these equations, the probabilistic index term weights $P(d_m \epsilon K_{ij} | tf_{im}=l)$ can be derived. The parameters π_{ij} and λ_{ij} can be estimated without feedback information from the document collection.
Experimental evaluations of this model were only partially successful. In [Harter 75a, Harter 75b], the χ^2-test rejected the hypothesis of a 2-Poisson distribution for 62 % of the terms tested. Experiments with a higher number of classes (termed n-Poisson model) as described in [Srinivasan 90] also did not give clear improvements. In the study [Margulis 91], an improved parameter estimation method is applied in combination with longer documents than in previous evaluations, thus leading to the result that the assumption of an n-Poisson distribution holds for about 70% of all terms.

6.3.5 Retrieval with probabilistic indexing

Having computed probabilistic indexing weights either by means of the 2-Poisson model or the Darmstadt Indexing Approach, these weights can be used for improving retrieval quality in comparison to approaches based on binary indexing. As a simple retrieval function for this purpose, the BII model can be applied (see e.g. [Fuhr & Buckley 91] for experimental results). However, this model does not allow for a weighting of search terms w.r.t. the query. So we look for a model that combines both kinds of weighting.

6.3.5.1 A linear decision-theoretic retrieval function

Let u_{mi} denote the indexing weight of term t_i w.r.t. document d_m and c_{ki} the query term weight of t_i in query q_k. Then, with q_k^T denoting the set of query terms and d_m^T the set of document terms, one can consider the scalar product as retrieval function with

$$\varrho(q_k^D, d_m^D) = \sum_{t_i \in q_k^T \cap d_m^T} c_{ki} \cdot u_{mi}.$$

As mentioned in [Wong & Yao 89], this retrieval function can be given a utility theoretic interpretation in the case of probabilistic indexing weights u_{mi}: The weight c_{ki} can be regarded as the utility of the term t_i, and the retrieval function gives the expected utility of the document with respect to the query. So this function does not estimate the probability of relevance (or a value that is a monotonic function of this probability); however, it is a utility-theoretic function, and the justification of the PRP is also utility-theoretic. On the other hand, in this function, the query term weights c_{ki} cannot be estimated by means of a probability estimation procedure (see [Fuhr 89a]); so these weights either have to be specified explicitly by the user, or they have to be derived by some ad-hoc procedure (see e.g. [Fuhr & Buckley 91]).

6.3.5.2 The RPI model

The retrieval-with-probabilistic-indexing (RPI) model described in [Fuhr 89a] is a model especially developed for combining probabilistic indexing weights with search term weighting based on relevance feedback.
In order to incorporate probabilistic indexing in this model, we assume a fixed number of binary indexings per document. This leads to an extended event space $\underline{Q} \times \underline{D} \times I$, where I denotes a set of indexers who produce a binary indexing (for every document in the collection) each. In this event space, a single event is a query-document pair with a relevance judgement, a specific (binary) indexing and a set of relevance descriptions for the terms w.r.t. the document.
Now we describe the representations and descriptions of documents and queries used in the RPI model. The query representations and descriptions are similar to those of the BIR model: a query representation q_k is a pair (q_k^T, q_k^J), where q_k^T denotes the set of query terms and q_k^J is the set of relevance judgements with $q_k^J = \{(\underline{d_m}, r(\underline{d_m}, \underline{q_k}))\}$; a query description q_k^D is a set of query terms with associated weights, where the weights are slightly different to those of the BIR model (see below). Instead of document representations, we regard representations of (document,indexer) pairs here. Such a pair is represented by a pair of vectors $d_m = (\vec{d}_m, \vec{c}_m)$ of relevance descriptions and assignment decisions, each for the whole set of index terms. So $\vec{d}_m$ is the vector $(d_{m_1}, \ldots, d_{m_n})^T$, where d_{m_i} is the relevance description of t_i w.r.t. d_m, and $\vec{c}_m$ is the binary vector $(c_{m_1}, \ldots, c_{m_n})^T$ with $c_{m_i} = C_i$, if t_i has been assigned to d_m and $c_{m_i} = \bar{C}_i$ otherwise. The document description d_m^D used here is a set of terms with indexing weights.
Let $\vec{x} = (x_1, \ldots, x_n)$ denote a set of relevance descriptions for the index terms $t_1, \ldots t_n$. The RPI model now aims at the estimation of the probability $P(R|q_k, \vec{x})$ that a document with relevance descriptions $\vec{x}$ is relevant w.r.t. q_k.
By applying Bayes' theorem, we get

$$O(R|q_k, \vec{x}) = O(R|q_k) \frac{P(\vec{x}|R, q_k)}{P(\vec{x}|\bar{R}, q_k)}. \tag{6.12}$$

The linked dependence assumption

$$\frac{P(\vec{x}|R, q_k)}{P(\vec{x}|\bar{R}, q_k)} = \prod_{i=1}^{n} \frac{P(x_i|R, q_k)}{P(x_i|\bar{R}, q_k)} \tag{6.13}$$

yields

$$O(R|q_k, \vec{x}) = O(R|q_k) \prod_{i=1}^{n} \frac{P(x_i|R, q_k)}{P(x_i|\bar{R}, q_k)}. \tag{6.14}$$

Now two assumptions are made:

1. The relevance description x_i of a term t_i depends only on the correctness of t_i, it is independent of the correctness of other terms and relevance.

2. The correctness of a term (w.r.t. a document) depends only on relevance, it is independent of the correctness of other terms.

With these assumptions, we get

$$O(R|q_k,\vec{x}) = O(R|q_k)\prod_{i=1}^{n}\frac{P(x_i|C_i)\cdot P(C_i|R,q_k)+P(x_i|\bar{C}_i)\cdot P(\bar{C}_i|R,q_k)}{P(x_i|C_i)\cdot P(C_i|\bar{R},q_k)+P(x_i|\bar{C}_i)\cdot P(\bar{C}_i|\bar{R},q_k)} \quad (6.15)$$

$$= O(R|q_k)\prod_{i=1}^{n}\frac{\frac{P(C_i|x_i)}{P(C_i)}\cdot P(C_i|R,q_k)+\frac{P(\bar{C}_i|x_i)}{P(\bar{C}_i)}\cdot P(\bar{C}_i|R,q_k)}{\frac{P(C_i|x_i)}{P(C_i)}\cdot P(C_i|\bar{R},q_k)+\frac{P(\bar{C}_i|x_i)}{P(\bar{C}_i)}\cdot P(\bar{C}_i|\bar{R},q_k)} \quad (6.16)$$

Here $P(C_i|x_i{=}d_{m_i})$ is the probabilistic indexing weight of t_i w.r.t. d_m, the probability that an arbitrary indexer assigned t_i to d_m. $P(C_i)$ is the probability that an arbitrary indexer assigned term t_i to an arbitrary document. $P(C_i|R,q_k)$ is the probability that an arbitrary indexer assigned t_i to an arbitrary document, given that it is relevant to q_k, and $P(C_i|\bar{R},q_k)$ is the corresponding probability for the nonrelevant documents. In the following, let $u_{m_i} = P(C_i|x_i{=}d_{m_i})$, $q_i = P(C_i)$, $p_{ik} = P(C_i|R,q_k)$ and $r_{ik} = P(C_i|\bar{R},q_k)$. Assuming that $P(C_i|R,q_k) = P(C_i|\bar{R},q_k)$ for all $t_i \notin q_k^T$, we get the final RPI retrieval function

$$O(R|q_k,\vec{x}{=}\vec{d}_m) = O(R|q_k)\prod_{t_i\in q_k^T}\frac{\frac{u_{m_i}}{q_i}p_{ik}+\frac{1-u_{m_i}}{1-q_i}(1-p_{ik})}{\frac{u_{m_i}}{q_i}r_{ik}+\frac{1-u_{m_i}}{1-q_i}(1-r_{ik})}. \quad (6.17)$$

If we assume that $P(C_i|\bar{R},q_k) \approx P(C_i)$, the approximate RPI formula from [Fuhr 89a] is derived:

$$O(R|q_k,\vec{x}{=}\vec{d}_m) \approx O(R|q_k)\prod_{t_i\in q_k^T}\frac{u_{m_i}}{q_i}p_{ik}+\frac{1-u_{m_i}}{1-q_i}(1-p_{ik}). \quad (6.18)$$

The parameters for the RPI formulas can be estimated by means of relevance feedback: let $\underline{D}_k^R$ denote the set of documents judged relevant w.r.t. q_k and $\underline{D}_k^N$ the set of nonrelevant documents. Then we can estimate q_i as the expectation of the indexing weight of t_i in an arbitrary document, p_{ik} as the corresponding expectation for the relevant documents and q_{ik} as the expectation for the nonrelevant documents:

$$q_i = \frac{1}{|\underline{D}|}\sum_{\underline{d}_j\epsilon\underline{D}} u_{j_i},$$

$$p_{ik} = \frac{1}{|\underline{D}_k^R|}\sum_{\underline{d}_j\epsilon\underline{D}_k^R} u_{j_i},$$

$$r_{ik} = \frac{1}{|\underline{D}_k^N|}\sum_{\underline{d}_j\epsilon\underline{D}_k^N} u_{j_i}.$$

6.4 IR as uncertain inference

Although the relevance models described in the previous sections have been rather successful in the past, there are three major shortcomings of this approach:

- The concept of relevance can be interpreted in different ways. One can either regard relevance of a document w.r.t. a query or information need, in which cases the user who submitted the query gives the relevance judgement; this approach has been taken so far this paper. Alternatively, relevance can be defined w.r.t. the query formulation, assuming that an objective judgement (e.g. given by specialists of the subject field) can be made. Of course, the latter approach would be more desirable in order to collect "objective" knowledge within an IR system.
- The relevance models are strongly collection-dependent, that is, all the parameters of a model are only valid for the current collection. When a new collection is set up, the "knowledge" from other collections cannot be transferred.
- Relevance models are restricted to rather simple forms of inference. In the models presented here, only the relationships between terms and queries are considered. It would be desirable to include information from other knowledge sources (e.g. from a thesaurus) in an IR model. With description-oriented approaches, this problem can partially be solved (see e.g. [Fuhr 89b]), but there is a need for a general model dealing with this issue.

In [Rijsbergen 86], a new paradigm for probabilistic IR is introduced: IR is interpreted as uncertain inference. This approach can be regarded as a generalization of the logical view on databases, where queries and database contents are treated as logical formulas. Then, for processing a query, those items from the database that imply the query have to be computed (see e.g. [**?**]). For document retrieval, this means that a document d_m is an answer to a query q_k if the query can be proven from the document, that is, if the logical formula $q_k \leftarrow d_m$ can be shown to be true. In order to satisfy this formula, additional knowledge not explicitly contained in the document can be used. For example, if d_1 is about 'squares', and q_1 asks for documents about 'rectangles', then the inference process can use the formula 'rectangle' $\leftarrow$ 'squares' in order to prove $q_1 \leftarrow d_1$. For IR, however, the logical database approach is not sufficient, since we have to cope with the intrinsic uncertainty of IR. Thus, we have to use uncertain inference. Rijsbergen proposes to use probabilistic inference for this purpose. So probabilistic IR can be interpreted as estimating the probability

$$P(d \rightarrow q)$$

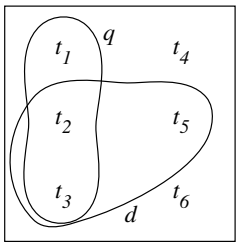
that the document implies the query. Rijsbergen also shows that the this implication should not be interpreted in the traditional sense, i.e. $d \rightarrow q = \neg d \vee q$; rather, it stands for the conditional probability of q given d:

$$P(d \rightarrow q) = P(q|d) \tag{6.19}$$

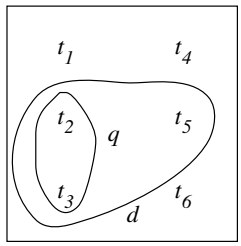
(As a numeric example, assume that $P(d) = 0.2$ and $P(q \wedge d) = 0.1$, where we get $P(q|d) = 0.5$, but $P(\neg d \vee q) = 0.9$.) E.g., consider a query about 'rectangles' and a document about 'quadrangles'. Since there is a certain probability that this document also may be an answer to the query, we might have an uncertain rule in our knowledge base, stating that 'quadrangles' implies 'rectangles' with a certain probability. Thus we could show that the document implies the query with the corresponding probability.
In this framework, the concept of relevance does not feature. The most obvious way for mapping the outcome of the uncertain inference process onto the probability of relevance is via another conditional probability:

$$P(R|q,d) = P(R|q \leftarrow d)P(q \leftarrow d) + P(R|\neg(q \leftarrow d))P(\neg(q \leftarrow d)) \tag{6.20}$$

This leaves us with the problem of estimating the probabilities $P(R|q \leftarrow d)$ and $P(R|\neg(q \leftarrow d))$. So far, there is no obvious method for deriving the values of these parameters. On the other hand, according to formula (6.20), $P(R)$ is a monotonuous function of $P(q \leftarrow d)$, thus only the value of the latter probability is required for a ranking of documents w.r.t. to a query.
In [Nie 89], Rijsbergen's approach is extended by arguing that besider the implication $d \rightarrow q$, also the inverse implication $d \leftarrow q$ should be considered. As a simple example, consider a query and two documents d and d' yielding the same probability of implying the query. However, d may be a short article and d' a whole book. In this case, most users would prefer the article. Nie terms the implication $d \rightarrow q$ as precision-oriented inference and $d \leftarrow q$ as recall-oriented inference.



$P(d \rightarrow q) = 2/3$
$P(q \rightarrow d) = 2/3$



$P(d \rightarrow q) = 2/3$
$P(q \rightarrow d) = 1$

Figure 6.5: Precision- vs. recall-oriented inference

As another example, consider figure 6.5. Assuming that terms are disjoint and have equal probability, we see that in both cases $P(d \rightarrow q)$ is the same. However, in the right-hand case, the document is fully contained in the query. Thus, we would prefer this document over the other one. So Nie argues that the probability of relevance should be a function of the probability of both implications:

$$P(R|q,d) = f(P(d \rightarrow q), P(q \rightarrow d)) \tag{6.21}$$

However, so far no sound theoretical framework for determining this function exists.

6.4.1 Imaging

Since relevance does not feature explicitly in the uncertain inference approach, we face the problem of estimating the required probabilties. As a method for accomplishing this task, Rijsbergen proposes to use imaging.
In order to cope with the problem that documents in a collection may contradict each other, one can identify each document with a possible world W, that is, a set of propositions with associated truth values. Let τ denote a truth function, then $\tau(W, x)$ denotes the truth of the proposition x in the world W, where $\tau(W, x) = 1$ if x is true at W and $\tau(W, x) = 0$ if x is false at W.
Let x and y denote propositions. Then the probability of the conditional $y \rightarrow x$ is computed in the following way: Let $\sigma(W, y)$ denote the world most similar to W where y is true. Then $y \rightarrow x$ is true at W if and only if x is true at $\sigma(W, y)$.
For estimating $P(y \rightarrow x)$ (independent of a specific world), all possible worlds must be regarded. Each world W has a probability $P(W)$ so that they sum to unity over all possible worlds. Then $P(y \rightarrow x)$ can be computed in the following way:

$$\begin{aligned} P(y \rightarrow x) &= \sum_W P(W)\tau(W, y \rightarrow x) \\ &= \sum_W P(W)\tau(\sigma(W, y), y \rightarrow x) \\ &= \sum_W P(W)\tau(\sigma(W, y), x) \end{aligned} \tag{6.22}$$

So we have to sum over all possible worlds, look for the closest world where y is true, and add the truth of x for this world. This formula can be illustrated by an example shown in table 6.7. If we assume that $P(W_i) = 0.1$ for $i = 1, \ldots, 10$, then we get $P(y \rightarrow x) = 0.6$

6.4.2 Inference networks

When IR is regarded as uncertain inference as described above, then the structure of inference from documents to queries becomes more complex as in the case of the relevance models. In general, one gets an inference network. As a probabilistic formalism for inference networks with uncertainty, Bayesian inference networks have been described in [Pearl 88]. Turtle and Croft [Turtle & Croft 90] applied this formalism to document retrieval. An example inference network is shown in figure 6.6. Here each node representing either a document, a query or a concept can take on the value *true* or *false*. In contrast to the models discussed so far in this paper, we assume that there are two kinds of concepts, namely document concepts t_i and query concepts r_i.

W_i	$\tau(W_i, y)$	$\tau(W_i, x)$	$\sigma(W_i, y)$	$\tau(\sigma(W_i, y), x)$
1	0	1	2	0
2	1	0	2	0
3	0	0	6	1
4	0	1	6	1
5	0	0	6	1
6	1	1	6	1
7	0	0	8	1
8	1	1	8	1
9	0	0	10	0
10	1	0	10	0

Table 6.7: Imaging example

The directed arcs of this network indicate probabilistic dependence of nodes. The probability of a node being true depends only on the values of its parents. This relationship must be specified as a function within the node. In order to estimate the probability of $P(d \rightarrow q)$, the document node is set to *true* and then the probabilities of the depending nodes are computed until the value of $P(q = true)$ is derived.
Depending on the combining function of a node, an inference network may contain different types of nodes, e.g. for Boolean connectors as well as for probabilistic correlation as in the relevance models described before. As a simple example, assume that the representation concept r_1 is 'IR' which is defined as an OR-combination of the two document concepts (terms) t_1 = 'information retrieval' and t_2 = 'document retrieval'. Then the probability of r_1 being true can be computed by the function

$$P(r_1{=}true) = 1 - (1 - P(t_1{=}true))(1 - P(t_2{=}true)).$$

This approach has several advantages in comparison to the relevance models:

- As most probabilistic IR models can be mapped onto an inference network, this formalism gives a unifying representation for the different models (see e.g. [Turtle & Croft 90]). In contrast to these models, the network approach does not require the derivation of a closed probabilistic formula, so more complex interdependencies can be incorporated.
- The network approach allows for the combination of multiple sources of evidence. For example, information about the similarity of documents can be considered as well as knowledge from external sources like a thesaurus.
- Different query formulations and types of query formulations can be combined in order to answer a single query. For example, a Boolean and a probabilistic formulation can be used in parallel, where the network combines the results of both formulations.

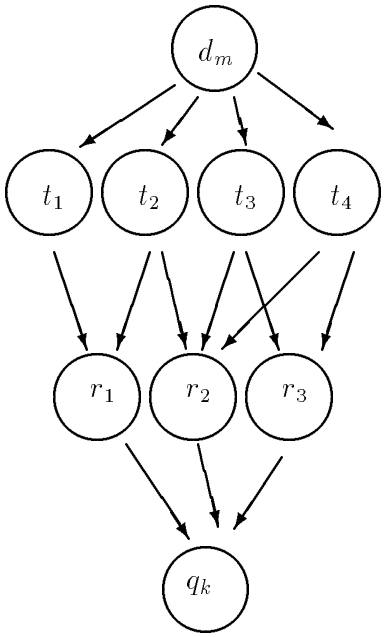


Figure 6.6: Example inference network

The major problem with the inference network approach, however, is the estimation of the arc weights when non-Boolean nodes are used. In this case, for example with query-specific relevance feedback, relevance models offer a better solution.

6.5 General problems

In this section, we discuss some problems that are general to any probabilistic IR model, namely parameter estimation, query expansion, and the representation of documents and queries.

6.5.1 Parameter estimation

6.5.1.1 Parameter estimation and IR models

The **parameter estimation procedure** used for the application of a model is in large parts independent of the tested model itself. So any experimental evaluation of IR models does not only compare the models itself, it compares the combination of IR models with specific parameter estimation procedures. Here we want to subdivide the whole estimation procedure into two parts: the estimation sample selection and the estimation method which uses the data from the sample in order to estimate the required parameters. The latter is the main topic of this paper and will be discussed

in detail in the following sections.
The selection of the estimation sample can be done in principle in two different ways: Following the definitions and assumptions of the model to be applied, in most cases it would be necessary to have a random sample of documents from which the parameters are to be derived. But this approach would require too large numbers of documents to be selected in order to provide any valuable information for the estimation method. So the only way out is to use some kind of best-first selection method, that is to apply an initial ranking formula and collect feedback information from the top ranking documents. This means that instead of $P(t_i|q_k, r)$, the probability that a term t_i occurs in an arbitrary document judged relevant/non-relevant w.r.t. request q_k, we estimate $P(t_i|q_k, r, sel)$, the probability that t_i occurs in an arbitrary document with judgement r which is selected by the initial ranking function. As most of the ranking formulas used for the initial ranking (e.g. coordination level match or inverse document frequency weights) prefer documents which contain the query terms, it is obvious that these estimates in general will be higher than the probabilities $P(t_i|q_k, r)$ for which they are used. As long as this problem has not been solved, all experimental results for more sophisticated models are of preliminary nature.
Despite of the problems described above, we will follow the assumption of random estimation samples in the following.

6.5.1.2 Standard methods of parameter estimation

Having described some of the peculiarities with parameter estimation for probabilistic IR models, we will now give a definition for an optimum parameter estimate and describe a method how this estimate can be achieved. For that, we will first give a more formal description of the parameter estimation problem.
The general situation is as follows: in a collection of documents, each document may have several features e_i. For a fixed set of n feature pairs, we are seeking for estimates of $P(e_i|e_j)$, the probability that a random document has feature e_i, given that it has feature e_j. In a random sample of g objects, we observe f objects with feature e_j, of which h objects also have the feature e_i. In the case of the BIR model, the features e_j are either relevance or non-relevance w.r.t. the current query, and the features e_i denote the presence of the terms t_i.
Now the problem is to derive an estimate $p(e_i|e_j, (h, f, g))$ for $P(e_i|e_j)$, given the parameter tripel (h, f, g). The most simple estimation method uses the **maximum likelihood estimate**, which yields $p(e_i|e_j, (h, f, g)) = h/f$. Besides the problem with the quotient $0/0$, this estimate also bears a bias (see the experimental results in [Fuhr & Hüther 89]).
Bayesian estimates are preferred most in IR research. This method assumes that we have previous knowledge about the parameter Q to be estimated. Based on the knowledge of the prior distribution of this parameter, we use the sample data in

order to derive the posterior distribution of the parameter: Assume that X is a random variable which can have discrete values $x_1, x_2, \ldots$ depending on parameter Q which is a continuous random variable (it is also possible to assume Q as a discrete variable, but all applications described in the following assume a continuous one). Then $P(X{=}x_k|Q{=}q)$ is the probability that X will take the value x_k given that parameter Q has the value q. With $f(q)$ describing the prior distribution of parameter Q we get the posterior distribution

$$g(q|x_k) = \frac{f(q) \cdot P(X{=}x_k|Q{=}q)}{\int_{-\infty}^{\infty} f(q) \cdot P(X{=}x_k|Q{=}q)dq} \tag{6.23}$$

Further methods have to be applied in order to derive an estimate for q from this formula.

In the following discussion, we will restrict to a specific application of the Bayesian method to our problem where a beta distribution is assumed as prior distribution. The density of the beta distribution is given by the formula

$$f(p) = \frac{1}{B(a,b)}\, p^{a-1}(1-p)^{b-1}$$

with $B(a,b){=}\Gamma(a) \cdot \Gamma(b)/\Gamma(a+b)$ and $a, b > 0$ are parameters to be chosen. The assumption of a beta distribution is made (explicitly or implicitly) for most applications of the Bayesian method in IR ([Robertson & Sparck Jones 76], [Rijsbergen 77], [Bookstein 83a], [Losee 88]). In contrast to our approach, these authors assume a single (beta) distribution for the parameters p_{ij}, independently of the probabilities p_j. Furthermore, our optimum estimate also depends on the collection size g, while the sequential learning model described in [Bookstein 83a] assumes that the prior distribution is independent of g. With the beta distribution as prior and the fact that we have observed the frequencies (h, f), we get:

$$g(p|\tfrac{h}{f}) = \frac{p^{a-1}(1-p)^{b-1}\binom{f}{h}p^h(1-p)^{f-h}}{\int_0^1 p^{a-1}(1-p)^{(b-1)}\binom{f}{h}p^h(1-p)^{f-h}dp}$$

Using the relationship $B(a,b) = \int_0^1 p^{a-1}(1-p)^{b-1}dp$, we get as posterior distribution:

$$g(p|\tfrac{h}{f}) = \frac{p^{h+a-1}(1-p)^{f-h+b-1}}{B(h+a, f-h+b)} \tag{6.24}$$

From this distribution, different estimates can be derived. One possibility is to choose that value p_{max} for which $g(p|\frac{h}{f})$ takes its maximum value. This approach is quite similar to the maximum likelihood method. With

$$\frac{dg(p|\frac{h}{f})}{dp} \stackrel{!}{=} 0$$

we get

$$p_{max} = \frac{h + a - 1}{f + a + b - 2}$$

A second approach is based on the definition of a loss function. Besides the well-known function

$$L_1(\hat{p}, p_{ij}) = (\hat{p} - p_{ij})^2, \tag{6.25}$$

we also regard the loss function

$$L_2(p, \hat{p}) = \frac{(p - \hat{p})^2}{p(1 - p)} \tag{6.26}$$

discussed in [Rijsbergen 77].

Now we seek for estimates p_L minimizing the expectation of the loss function, that is

$$\frac{d}{dp_L} \int_0^1 L(p, p_L) g(p) dp \stackrel{!}{=} 0$$

This yields the estimates[2]

$$p_{L_1} = \frac{h + a}{f + a + b}$$

and

$$p_{L_2} = \frac{h + a - 1}{f + a + b - 2} = p_{max}$$

Finally, in [Bookstein 83a] a proposal is made to use the expectation of p as estimate:

$$p_E(e_i|e_j, \tfrac{h}{f}) = \int_0^1 g(p|\tfrac{h}{f})\ p\ dp$$

For the beta prior, we get

$$p_E(e_i|e_j, \tfrac{h}{f}) = \frac{h + a}{f + a + b} = p_{L_2}$$

It is interesting to notice that the four different methods for deriving an estimate from be posterior distribution yield only two different results when a beta distribution is assumed as prior. In any case there is still the problem of the estimation of the parameters a and b (see also the following section). In [Losee 88] several heuristic strategies for the choice of these parameters are evaluated. [Robertson & Sparck Jones 76] assumed a=b=$\frac{1}{2}$ in their experiments (following a proposal in [Cox 70]) and in [Robertson 86] parameter combinations with $a + b$=1 are discussed (according to our definition of p_{L_1}). For a=b=1 the beta distribution is identical with the uniform distribution. In this case p_{L_2}=p_{max} yields the same estimate as the maximum likelihood method.

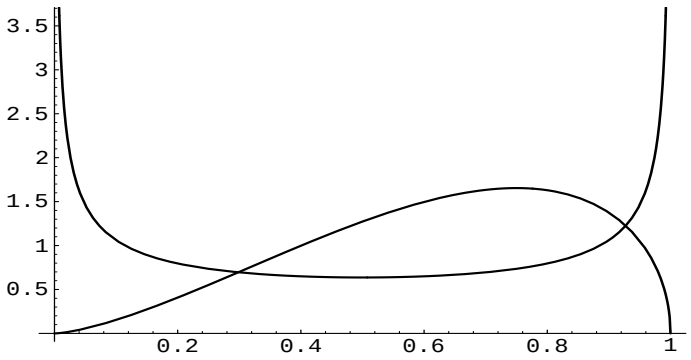


Figure 6.7: Prior (upper curve) and posterior distribution for $a = b = 0.5$, $f = 3$, $h = 2$

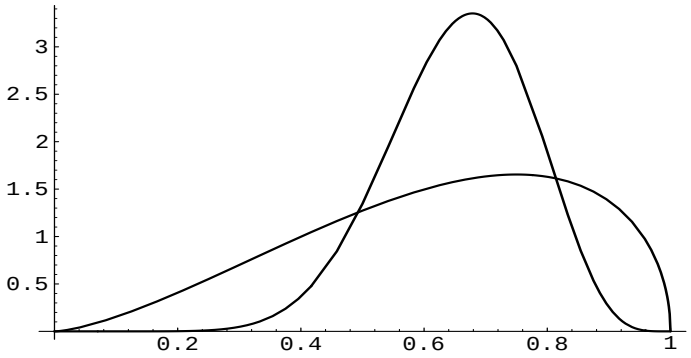


Figure 6.8: Posterior distributions for $a = b = 0.5$, $f = 3$, $h = 2$ and $f' = 15$, $h' = 10$

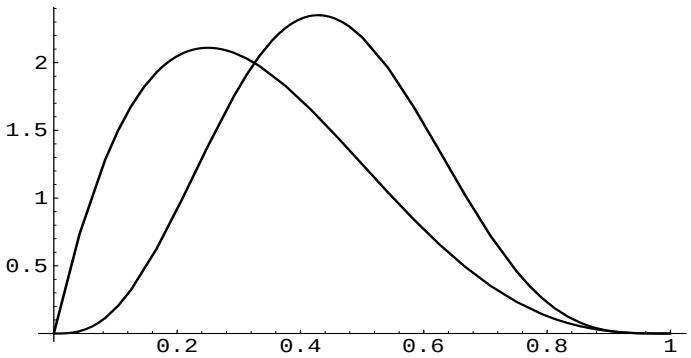


Figure 6.9: Prior (left curve) and posterior distribution for $a = 2$, $b = 4$, $f = 3$, $h = 2$

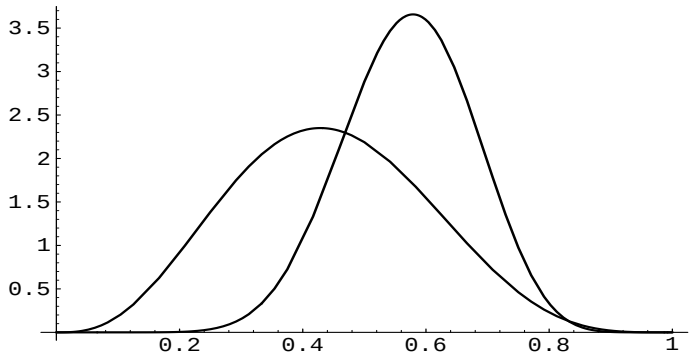


Figure 6.10: Posterior distributions for $a = 2$, $b = 4$, $f = 3$, $h = 2$ and $f' = 15$, $h' = 10$

In order to illustrate this approach, we give some examples of prior and posterior distributions for p_{L_2}. Figure 6.7 shows prior and posterior distributions with $a = b = 0.5$ for $f = 3$ and $h = 2$. That the degree of influence of the prior distribution depends on the size of the learning sample can be seen from figure 6.8, where two posterior distributions with the same a and b parameters, but one for $f = 3$ and $h = 2$ and the other for $f' = 15$ and $h' = 10$ are given. Since the Beta function with $a = b = 0.5$ (see figure 6.7) has its minimum at 0.5 and is converging to infinity for $x = 0$ or $x = 1$, it may not meet the real data. A more intuitive function would be a convex, non-symmetric function with a maximum close to 0. As an example of such a function, we have chosen $a = 2$ and $b = 4$ in the following two figures. Figure 6.9 shows prior and posterior distributions $f = 3$ and $h = 2$ and figure 6.10 gives the two posterior distributions for $f = 3$ and $h = 2$ and for $f' = 15$ and $h' = 10$, respectively.

6.5.1.3 Optimum parameter estimation

Experimental results described in [Fuhr & Hüther 89] have shown that the assumption of a beta prior may be theoretically inadequate. Instead, an optimum estimate based on empirical distributions is derived: Our aim is to derive an optimum estimate $p_{opt}(e_i, e_j)$ for a feature pair with the parameter triplet (h, f, g). We are regarding a total of n pairs, where $Z(h, f, g)$ is the set of feature pairs with (h, f, g) and $\sum_{h,f} |Z(h, f, g)| = n$. Based on this information, we want to derive a point estimate for $P(e_i|e_j)$, which is required for all known probabilistic IR models. We will justify our choice of $p_{opt}(e_i|e_j)$ by means of a loss function. First, we introduce the following notations: Let p_{kl}=$P(e_k|e_l)$ and p_l=$P(e_l)$ denote the probability that a random object has feature e_l. Furthermore assume that Q is a random variable of the prior distribution of the p_{kl}'s. In contrast to the approaches described in the previous section, no specific assumption about this prior distribution is made. Finally let Z_g be

[2] In Rijsbergen's paper, a false value (that of p_{L_1}) is given as result for p_{L_2}

the random variable for the frequency pairs (h, f) we observed within the g objects, such that $P(Z_g{=}(h,f)|Q{=}p_{kl})$ gives us the probability that a feature pair (e_k, e_l) with underlying probability p_{kl} has the frequencies (h, f) within the g objects.
Now an estimate p_{min} is chosen such that the expected value of L_1 (see eqn 6.25) is minimized:

$$\begin{aligned} p_{opt}(e_i|e_j,(h,f,g)) &= \min_{0\le\hat{p}\le 1}(E(L_1(\hat{p},p_{ij})) \\ &= \min_{0\le\hat{p}\le 1}\left(\sum_{(k,l)}(\hat{p}-p_{kl})^2 p_l P(Z_g{=}(h,f)|Q{=}p_{kl})\right) \end{aligned}$$

$$\frac{dE}{d\hat{p}} = \sum_{(k,l)} 2\hat{p}p_l P(Z_g{=}(h,f)|Q{=}p_{kl}) - \sum_{(k,l)} 2p_{kl}p_l P(Z_g{=}(h,f)|Q{=}p_{kl}) \quad (6.27)$$

$$\frac{dE}{d\hat{p}} \stackrel{!}{=} 0 \Longrightarrow$$

$$p_{opt} = \frac{\sum_{(k,l)} p_{kl}p_l P(Z_g{=}(h,f)|Q{=}p_{kl})}{\sum_{(k,l)} p_l P(Z_g{=}(h,f)|Q{=}p_{kl})} \quad (6.28)$$

(6.29)

Having justified our choice of p_{opt} this way, we now want to show how p_{opt} can be estimated on the basis of data from our learning sample of size g.
Therefore we define $E^+(h,f,g)$ as the numerator of eqn 6.28, i.e.

$$E^+(h,f,g) = \sum_{(k,l)} p_{kl} \cdot p_l \cdot P(Z_g{=}(h,f)|Q{=}p_{kl})$$

(the expected number of occurrences of (e_k, e_l)) and

$$E^-(h,f,g) = \sum_{(k,l)} (1-p_{kl}) \cdot p_l \cdot P(Z_g{=}(h,f)|Q{=}p_{kl})$$

(the expected number of occurrences of e_l without e_k), so that we get

$$p_{opt}(e_i|e_j,(h,f,g)) = \frac{E^+(h,f,g)}{E^+(h,f,g)+E^-(h,f,g)} \quad (6.30)$$

The expectations $E^+(h,f,g)$ and $E^-(h,f,g)$ can be approximated by the expected values $E(h,f,g)$ of the frequency distribution $|Z(h,f,g)|$:

$$E^+(h,f,g) = \sum_{(k,l)} \binom{g}{f} p_l^f (1-p_l)^{g-f} \binom{f}{h} p_{kl}^h (1-p_{kl})^{f-h} \cdot p_l \cdot p_{kl}$$

$$
\begin{aligned}
&= \frac{h+1}{g+1} \sum_{(k,l)} \left(\begin{array}{c} g+1 \\ f+1 \end{array} \right) p_l^{f+1} (1-p_l)^{g+1-(f+1)} \cdot \\
&\quad \left(\begin{array}{c} f+1 \\ h+1 \end{array} \right) p_{kl}^{h+1} \cdot (1-p_{kl})^{f+1-(h+1)} \\
&= \frac{h+1}{g+1} E(h+1, f+1, g+1) \\
&\approx \frac{h+1}{g} E(h+1, f+1, g) \qquad (6.31)
\end{aligned}
$$

The approximation used above is not critical, in comparison to the probability estimation the error is of second order.

$$
\begin{aligned}
E^-(h,f,g) &= \sum_{k,l} \left(\begin{array}{c} g \\ f \end{array} \right) p_l^f (1-p_l)^{g-f} \left(\begin{array}{c} f \\ h \end{array} \right) p_{kl}^h (1-p_{kl})^{f-h} p_l \cdot (1-p_{kl}) \\
&= \frac{f+1-h}{g+1} \sum_{k,l} \left(\begin{array}{c} g+1 \\ f+1 \end{array} \right) p_l^{f+1} (1-p_l)^{g+1-(f+1)} \cdot \\
&\quad \left(\begin{array}{c} f+1 \\ h \end{array} \right) p_{kl}^h (1-p_{kl})^{f+1-h} \\
&= \frac{f+1-h}{g+1} E(h, f+1, g+1) \\
&\approx \frac{f+1-h}{g} E(h, f+1, g) \qquad (6.32)
\end{aligned}
$$

With these approximations for $E^+(h,f,g)$ and $E^-(h,f,g)$, we can estimate p_{opt} according to formula 6.30 as

$$
p_{opt}(e_i | e_j, (h,f,g)) \approx \frac{(h+1)\, E(h+1, f+1, g)}{(h+1)\, E(h+1, f+1, g) + (f+1-h) \cdot E(h, f+1, g)} \qquad (6.33)
$$

To apply this formula, we need a sufficient amount of data about our learning sample of size g (that is, we have to observe a large number of feature pairs (e_k, e_l)). Then we can use the numbers $|Z(h,f,g)|$ of the frequency distribution as approximation of the expected values $E(h,f,g)$. Experimental comparisons of this optimum estimate with Bayesian estimates showed almost no difference in terms of retrieval quality, whereas maximum likelihood estimates gave significantly worse results [Fuhr & Hüther 89].

6.5.2 Query expansion

Closely related to the problem of parameter estimation is the question: Which terms should be included in the query formulation? In derivation of the BIR model, we have assumed that $p_{ik} = q_{ik}$ for all terms not occurring in the query. Of course, there will be a number of additional terms for which this assumption does not hold, so they should be included in the query, too. If the query formulation is to be expanded by additional terms, there are two problems that are to be solved, namely

1. how are these terms selected and
2. how are the parameters c_{ik} estimated for these terms?

For the selection task, three different strategies have been proposed:

- dependent terms: Here terms that are dependent on the query terms are selected. For this purpose, the similarity between all terms of the document collection has to be computed first ([Rijsbergen et al. 81]).
- feedback terms: From the documents that have been judged by the user, the most significant terms (according to a measure that considers the distribution of a term within relevant and nonrelevant documents) are added to the query formulation ([Salton & Buckley 90]).
- interactive selection: By means of one of the methods mentioned before, a list of candidate terms is computed and presented to the user who makes the final decision which terms are to be included in the query ([Harman 88]).

With respect to the parameter estimation task, experimental results have indicated that the probabilistic parameters for the additional query terms should be estimated in a slightly different way than for the initial query terms, e.g. by choosing different paramaters a and b in the beta estimate (that is, a different prior distribution is assumed for the new terms) (see [Robertson 86, Salton & Buckley 90]).

The experimental results available so far indicate that the dependent terms method does not lead to better retrieval results [Rijsbergen et al. 81], whereas clear improvements are reported in [Salton & Buckley 90] and [Kwok 91] for the feedback terms method.

6.5.3 Representation of documents and queries

So far in this paper, we have assumed that a query is a set of terms, which in turn are words (with the exception of the Bayesian network approach, where a query also may contain Boolean connectors). But as more advanced text analysis methods are available, there is a growing need for models that can be combined with refined representation formalisms. Several authors have investigated the additional use of phrases as query terms (see e.g. [Croft 86], [Fagan 89] [Sembok & Rijsbergen 90]). The results from this experimental work do not give a clear indication whether retrieval

quality can be improved with phrases as query terms. However, three problems should be considered when interpreting these results:

1. Phrases are a different kind of terms in comparison to words. For this reason, the application of the standard weighting schemes developed for words may not be appropriate.

2. When phrases as well as their components are used as query terms, these terms are highly dependent, so a dependence model should be used, as for example in [Croft et al. 91].

3. The document collections used for experiments may be too small in order to show any benefit from phrases. Other experiments with larger collections have successfully used phrases in addition to words ([Biebricher et al. 88]), [Fuhr et al. 91]).

A dependence model for phrases is not sufficient, since this approach only regards the occurrence of the phrase components in a document, without considering the syntactical structure of the phrase as it occurs in the document. So the certainty of identification also should be considered (e.g. whether the components occur adjacent or only within the same paragraph). This can be achieved via application of probabilistic indexing methods (e.g. with the BII model in combination with a description-oriented approach); furthermore, with regard to the first point from above, indexing methods compute correct weights for different kinds of terms.

In contrast to the approaches discussed so far that are based on free text search, there is also work on automatic indexing with a controlled vocabulary (descriptors). In this case an indexing dictionary is required that contains pairs (with associated weights) of text terms and descriptors. For the problem of computing an indexing weight for a descriptor with indications from different terms within a document, either model-oriented ([Robertson & Harding 84]) or description-oriented approaches ([Biebricher et al. 88], [Fuhr et al. 91]) can be applied. Although many researchers are convinced that a controlled vocabulary offers no advantages over a free vocabulary, there is in fact little substantial experimental evidence supporting this position ([Salton 86]).

As an example for an advanced text representation method, in [Chiaramella & Nie 90] a semantic network representation is used for medical documents and queries (in conjunction with a fuzzy retrieval function). For probabilistic IR, this kind of representation is a challenge. Theoretically, the uncertain inference approach as developed by van Rijsbergen could be applied here — although the problem of parameter estimation has not been finally solved. Another possibility is the description-oriented approach described in [Fuhr 89b], which, however, requires fairly large learning samples for application.

6.6 Conclusion and outlook

In this chapter, the major concepts of probabilistic IR have been described. Following this goal, we have only occasionally referred to experimental results. Within the TREC initiative, a number of methods is tested with a 2 GB text database cintaining about 750,000 documents. First results are described in [Harman 93].
As new possible applications for IR methods arise, the scope of the field also has to be revised. In the past, collections with short documents (i.e. abstracts) have been investigated almost exclusively. Nowadays, fulltext document databases are set up for many applications; the TREC initiative mentioned above also addresses this point. With multimedia documents, there is the problem of representation for the non-textual parts: should they be represented by a set of keywords, or are structured descriptions required (as in [Rabitti & Savino 91])?
In the field of database research, there is also growing interest in methods for coping with imprecision in databases [IEEE 89, Motro 90]. As new databases for technical, scientific and office applications are set up, this issue becomes of increasing importance. A first probabilistic model that can handle both vague queries and imprecise data has been presented in [Fuhr 90]. Furthermore, the integration of text and fact retrieval will be a major issue (see e.g. [Rabitti & Savino 90]).
Finally, it should be mentioned that the models discussed here do scarcely take into account the special requirements of interactive retrieval. Even the feedback methods are more or less related to batch retrieval, where feedback from the first retrieval run is used in order to improve the quality of the second run (an exception must be made for the paper [Bookstein 83a], where iterative feedback methods are discussed). Since an interactive system allows a larger variety of interactions than just query formulation and relevance feedback (see e.g. [Croft & Thompson 87]), these interactions should be incorporated in a model for interactive probabilistic retrieval. Especially, the role of probabilistic parameters in the interaction process should be investigated: How should probabilistic weights (if at all) be presented to the user, and should there be a possibility for a user to specify probabilistic weights? In order to answer these questions, more experimental research is necessary. A major impediment for this kind of research is the fact that experiments with interactive systems and real users require a much bigger effort than testing ranking procedures in a batch environment.

Kapitel 7

Models based on predicate logic

7.1 Introduction

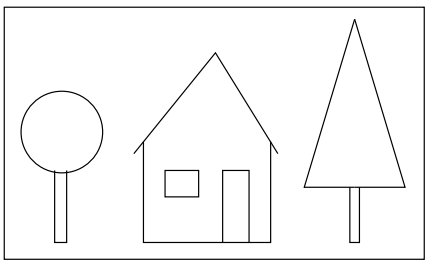


Abbildung 7.1: Example image

For advanced retrieval applications, e.g. multimedia retrieval, the classical IR models based on propositional logic are not sufficient.
As a simple example, consider image retrieval. For a picture like in figure 7.1, indexing according to classical IR models would only assign the terms `tree` and `house`. However, from this description, it does not become clear that there are two trees, one right and one left of the only house. Thus, users looking for images with two trees or with a tree on the left of a house could not express precisely their need, and thus they would get many incorrect answers. This problem can be overcome by using predicate logic. In our example, the document indexing could be:
`tree(t1). house(h1). tree(t2). left(t1, h1). left(h1,t2).`
Here `t1`, `t2` and `h1` are constants denoting the objects found in the picture. Let capital letters denote variables, then we can formulate a query looking for an image with two trees as follows:
`tree(X) & tree(Y) & X`$\neq$ `Y`
and searching for a tree on the left of a house is expressed as
`tree(X) & house(Y) & left(X,Y).`

In the following, we will describe two major approaches for IR models based on predicate logic, namely terminological logic and Datalog. For the latter, we also will present a probabilistic variant.

7.2 Terminological logic

7.2.1 Thesauri

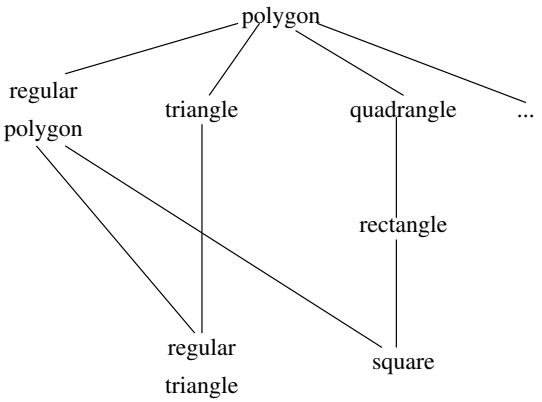


Abbildung 7.2: Thesaurus structure

If we look at classical thesauri, then we see that a structure like e.g. in picture 7.2 still can be expressed in propositional logic. For example, the fact that a square is a subconcept of both a quadrangle and a regular polygon can be expressed by means of the logical formula
`square` $\Leftrightarrow$ `quadrangle` $\wedge$ `regular-polygon`.
Terminological logics are based originally on semantic networks (like e.g. KL-ONE), but their semantics is clearer defined. In comparison to thesauri, terminological logics offer two major advantages in terms of expressiveness:

1. Since terminological logics are based on predicate logic, it is possible to name also instances of concepts in the description of a documents, like e.g. in the image example from above. Thus, we are able to distinguish between identical and different instances of concepts. In addition to monadic predicates representing concepts, dyadic predicates describe relationships between objects, e.g. that tree `t1` is left of house `h1`.
2. For describing the relationships between different concepts terminological logics offer a more powerful machinery than thesauri. For example, we can define a student as a person who is enrolled at a university.

7.2.2 Elements of terminological logic

The basic elements of terminological logic are concepts and roles. **Concepts** are monadic predicates like e.g. `person` and `document`. **Roles** are dyadic predicates like e.g. `author` (denoting authorship of a book) and `refers-to` (for referential links between two documents).

The relationships between concepts and roles are described by means of **terminological axioms**. An axiom can be either a connotation or a definition. A **connotation** gives only necessary conditions for a concept, e.g.
`man` $<\cdot$ `person`
only states that a man is a person, whereas a **definition** names necessary and sufficient conditions, like e.g.
`square =` (and `rectangle regular-polygon`)
informs the system that each object which is both a rectangle and a regular polygon also is a square.

Instances of concepts and roles are defined by means of **assertions**, e.g.
`document[d123]. person[Smith]. author[d123,Smith].`
names a document `d123`, a person `Smith` and tells the system that `Smith` is the author of `d123`.

In the following, we will describe a specific terminological logic called MIRTL (Multimedia IR terminological logic), as presented in [Meghini et al. 93].

A MIRTL knowledge base consists of a terminological and an assertional module. The **terminological module** contains concepts and roles along with definitions and connotations. Let M and C denote concepts and D and R denote roles, then a definition is an expression of the form M = C (or D = R, respectively), and a connotation has the form M $<\cdot$ C (or D $<\cdot$ R, respectively).

The **assertional module** consists of assertions of the form $C[i]$ or of the form $R[i_1, i_2]$, where C is a concept, R is a role and i, i_1 and i_2 are individual constants. For example, `document[d123]` and `person[Smith]` state that `d123` is a document and `Smith` is a person, and author[d123,Smith] stands for the fact that `Smith` is an author of `d123`.

The syntax for describing concepts and roles in the terminological module is as follows:

$$
\begin{array}{lcl}
\langle concept \rangle & ::= & \langle monadic\ predicate\ symbol \rangle \\
& & \mid (\textbf{top}) \\
& & \mid (\textbf{bottom}) \\
& & \mid (\textbf{a-not}\ \langle monadic\ predicate\ symbol \rangle) \\
& & \mid (\textbf{sing}\ \langle individual\ constant \rangle) \\
& & \mid (\textbf{and}\ \langle concept \rangle^{+}) \\
& & \mid (\textbf{all}\ \langle role \rangle\ \langle concept \rangle) \\
& & \mid (\textbf{c-some}\ \langle role \rangle\ \langle concept \rangle) \\
& & \mid (\textbf{atleast}\ \langle natural\ number \rangle\ \langle role \rangle) \\
& & \mid (\textbf{atmost}\ \langle natural\ number \rangle\ \langle role \rangle) \\
\langle role \rangle & ::= & \langle dyadic\ predicate\ symbol \rangle \\
& & \mid (\textbf{inv}\ \langle role \rangle)
\end{array}
$$

For explaining the meaning of these constructs, let the symbols C, C_1, C_2, ... stand for concepts and R, R_1, R_2, ... for roles.
(**and** $C_1\ C_2 \ldots C_n$) denotes the set of all individuals that belong at the same time to concept C_1 and C_2 and $\ldots C_n$. For example, we can state that a regular triangle is both a triangle and a regular polygon:
`regular-triangle = (and triangle regular-polygon).`
(**c-some** $R\ C$) denotes the set of those individuals having at least one R that is a C. Assume that a German paper is a paper with at least one German author, which can be expressed as
`german-paper = (and paper (c-some author german))`
(**all** $R\ C$) denotes the set of those individuals whose R's are all C's. As an example, assume that a student paper is a paper where all authors are students:
`student-paper = (and paper (all author student))`
(**a-not** M) denotes the set of all individuals of the domain that are not denoted by the concept M. For example, a non-German is a person who is not German:
`non-german = (and person (a-not german))`
(**top**) and (**bottom**) denote the set of all individuals of the domain of discourse and the empty set, respectively.
(**sing** i) denotes the concept containing only the individual denoted by i. This allows for using a single individual for the definition of further concepts, e.g.
`unido=(sing univ-dortmund)`
(**atleast** $n\ R$) (resp. (**atmost** $n\ R$)) denotes the set of those individuals having at least (resp. at most) n R's. For example, assume that a multilingual person is a person who speaks at least two languages:
`multilingual = (and person (atleast 2 speaks-lang))`
Chinese parents are allowed to have at most 1 child:
`chinese-parent = (and (chinese (atmost 1 child)))`

Finally, (**inv** R) denotes the set containing the inverses of those pairs denoted by R, e.g.
`wrote = (inv author).`
In addition to the basic syntax, we also use the following abbreviations:

$$\begin{aligned}(\textbf{exactly}\ n\ R) &\hat{=} (\textbf{and}\ (\textbf{atleast}\ n\ R)\ (\textbf{atmost}\ n\ R))\\ (\textbf{func}\ R\ C) &\hat{=} (\textbf{and}\ (\textbf{all}\ R\ C)\ (\textbf{exactly}\ 1\ R))\\ (\textbf{no}\ R) &\hat{=} (\textbf{atmost}\ 0\ R)\end{aligned}$$

For example, definig a student as a person who is enrolled at exctly one university can be expressed as follows:

```
student  = (and person  (atleast 1 enrolled)
                        (atmost 1 enrolled)
                        (all enrolled university))
         = (and person  (exactly 1 enrolled)
                        (all enrolled university))
         = (and person  (func enrolled university))
```

In a similar way, a bachelor can be defined as a man who has no spouse:
`bachelor` = (and `man` (no `spouse`))

7.2.3 Semantics of MIRTL

Now we specify the meaning of term constructors and other primitives of MIRTL.
An **interpretation** $\mathcal{I}$ over a nonempty set of individuals (domain) $\mathcal{D}$ is a function that maps individual constants into elements of $\mathcal{D}$ such that

- $i_1 \neq i_2 \Longrightarrow \mathcal{I}(i_1) \neq \mathcal{I}(i_2)$,
- concepts are mapped onto subsets of $\mathcal{D}$ and
- roles are mapped onto subsets of $\mathcal{D} \times \mathcal{D}$ abgebildet werden,

in such a way that:

$\mathcal{I}(\textbf{top}) = \mathcal{D}$
$\mathcal{I}(\textbf{bottom}) = \emptyset$
$\mathcal{I}(\textbf{a-not}\ M) = \mathcal{D} \setminus \mathcal{I}(M)$
$\mathcal{I}(\textbf{sing}\ i) = \{x \in \mathcal{D} \mid x = \mathcal{I}(i)\}$
$\mathcal{I}(\textbf{and}\ C_1\ C_2 \ldots C_n) = \mathcal{I}(C_1) \cap \mathcal{I}(C_2) \cap \ldots \cap \mathcal{I}(C_n)$
$\mathcal{I}(\textbf{all}\ R\ C) = \{x \in \mathcal{D} \mid \forall y : \langle x, y\rangle \in \mathcal{I}(R) \Rightarrow y \in \mathcal{I}(C)\}$
$\mathcal{I}(\textbf{c-some}\ R\ C) = \{x \in \mathcal{D} \mid \exists y : \langle x, y\rangle \in \mathcal{I}(R) \wedge y \in \mathcal{I}(C)\}$
$\mathcal{I}(\textbf{atleast}\ n\ R) = \{x \in \mathcal{D} \mid \| \{y \in \mathcal{D} \mid \langle x, y\rangle \in \mathcal{I}(R)\} \| \geq n\}$
$\mathcal{I}(\textbf{atmost}\ n\ R) = \{x \in \mathcal{D} \mid \| \{y \in \mathcal{D} \mid \langle x, y\rangle \in \mathcal{I}(R)\} \| \leq n\}$
$\mathcal{I}(\textbf{inv}\ R) = \{\langle x, y\rangle \in \mathcal{D} \times \mathcal{D} \mid \langle x, y\rangle \in \mathcal{I}(R)\}$

Now we turn to **terminological axioms**. In the following, let M denote a monadic predicate, C a concept, D a dyadic predicate and R a role. Then a connotation has the form $M <\cdot C$ (or $D <\cdot R$, respectively), and a definition is formulated as $M = C$ (or $D = R$, respectively). An interpretation $\mathcal{I}$ over $\mathcal{D}$ satisfies a connotation

- $M <\cdot C$ iff $\mathcal{I}(M) \subseteq \mathcal{I}(C)$,
- $D <\cdot R$ iff $\mathcal{I}(D) \subseteq \mathcal{I}(R)$,

and a definition

- $M = C$ iff $\mathcal{I}(M) = \mathcal{I}(C)$,
- $D = R$ iff $\mathcal{I}(D) = \mathcal{I}(R)$.

For describing the semantics of **assertions**, let C denote a concept, R a role and i, i_1 and i_2 individual constants. An interpretation $\mathcal{I}$ over a domain $\mathcal{D}$ satisfies an assertion

$C[i]$ iff $\mathcal{I}(i) \in \mathcal{I}(C)$

$R[i_1, i_2]$ iff $\langle \mathcal{I}(i_1), \mathcal{I}(i_2) \rangle \in \mathcal{I}(R)$

An assertion is valid iff it is satisfied by all interpretations.
A MIRTL **knowledge base** is a pair $\Omega = (\Delta, \Gamma)$, where Δ is a set of terminological axioms and Γ is a set of assertions. An interpretation $\mathcal{I}$ satisfies a knowledge base $\Omega = (\Delta, \Gamma)$ iff it satisfies all axioms in Δ and all assertions in Γ. In this case, $\mathcal{I}$ is called a model of Ω.

7.2.4 Retrieval with terminological logic

Now we show how MIRTL can be used for IR. In fact, there are two possibilities, where the first follows the more traditional AI-oriented view of a single knowledge base, whereas the second adopts the IR approach of documents a possible worlds and thus uses separate assertional modules for each document

7.2.4.1 Documents as facts in a uniform knowledge base

This approach is presented in the original paper [Meghini et al. 93], where MIRTL is proposed as a single representation language for modelling documents and terminological knowledge as well as for query formulation.
We use a running example from the original paper for illustrating this approach. First, we show how documents with external attributes, logical, layout and content structure can be modelled:

```
(and  paper
      (func appears-in (sing SIGIR93)))
      (all author (func affiliation (sing IEI-CNR)))
      (c-some author (sing Carlo-Meghini))
      (c-some author (sing Fabrizio-Sebastiani))
      (c-some author (sing Umberto-Straccia))
      (c-some author (sing Constantino-Thanos))
      (exactly 4 author))[paper666]

(and  (func typeset-with (sing LaTeX))
      (func format (sing double-column))
      (no figure)
      (no running-header)
      (no running-footer))[paper666]

(and  (exactly 1 abstract)
      (exactly 5 section)
      (exactly 1 bibliography)) [paper666]

bibliography [paper666,bib666]

(and  (func typeset-with (sing BibTeX))
      (func style (sing plain))
      (exactly 22 reference)) [bib666]

(and  (c-some dw (sing  Mirtl))
      (c-some dw (sing  syn666))
      (c-some dw (sing  sem666))
      (c-some dw (sing  alg666))
      (c-some dw (sing  terminological-logic
                        (c-some modeling-tool (sing IR))))) [paper666]

terminological-logic [Mirtl]
syntax [Mirtl,syn666]
semantics [Mirtl,sem666]
inferential-algorithm [Mirtl,alg666]
```

Queries in MIRTL are expressed as concepts, too. Then the inference algorithm seeks for concepts that are subsumed by the query concept and outputs all instances of these concepts. For exmple, the following query asks for papers authored by Thanos which deal with the semantics of terminological logics (obviously, `paper666` is an answer to this query):

```
(and  paper
      (c-some  author (sing Costantino-Thanos))
      (c-some  dw
               (c-some (inv semantics) terminological-logic)))
```

Now we define some terminological knowledge by specifying some properties of terminological logic and extensional logic:

```
terminological-logic = (and logic
                            (func syntax term-orientated-syntax)
                            (func semantics extensional-semantics))
extensional-logic = (and logic
                         (func semantics extensional-semantics))
```

From these definitions, it follows that each terminological logic also is an extensional logic. Thus, by using this terminological knowledge, the following query asking for papers dealing with the semantics of extensional logics can also retrieve `paper666`:

```
(and paper
     (c-some dw
             (c-some (inv semantics) extensional-logic)))
```

7.2.4.2 Documents as separate assertional modules

This approach is based on the same idea as Boolean retrieval, where documents do not occur explicitly. Rather, they can be regarded as possible worlds assigning truth values to propositions, where each index term represents a proposition; if a term is assigned to a document, then this proposition is true within the corresponding world, otherwise it is false (see e.g. [Rijsbergen 89]). In retrieval, we search for worlds making the whole query statement true.

The same approach can also be taken in terminological retrieval. Now documents are no longer instances (of a concept like `document`), they represent possible worlds. In each world, we have specific instances for concepts and roles. For retrieval, we search for possible worlds containing instances of the query concept. In terms of MIRTL, we represent each document as a separate assertional module. For example, when we search for a multilingual person, then document `d1` might state that `peter` is multilingual, and the system should return the document identification along with the instance `peter`. Now there may be another document `d2` containing the information that `peter` is a monolingual person. Thus, we have a contradiction between the two documents, but this does not matter, since each document represents a different world. However, without a possible world semantics, we would have a contradiction in our knowledge base, thus making this approach fail. In general, when we search for documents talking about certain subjects, we cannot search for instances of concepts and roles without considering their context, i.e. the document in which they occur. So terminological retrieval is implemented as terminological reasoning at the level of documents, that is locating documents containing instances of the concepts and roles involved in the query concept.

For further illustrating this approach, assume that we have a news database and we use MIRTL for representing the content of a news message. Since news are about

events, we define this concept as follows:

```
event < (and  thing
              (all event_date date)
              (all event_location place)
              (all event_topic topic)
              (all event_involved subject))
```

Now assume that we have a document about Clinton visiting chancellor Kohl in Bonn. Its content can be represented by means of the following assertions:

```
event[e01].
event_date[e01,1/11/95].
event_location[e01,Bonn].
event_involved[e01,Clinton].
event_involved[e01,Kohl]
event_topic[foreign-policy]
```

Now a query asking for events where Kohl and Clinton met can be expressed as:

```
(and  event
      (c-some event_involved (sing Kohl))
      (c-some event_involved (sing Clinton))
```

For retrieval, the system now has to search for a document in which an instance of this concept occurs. With additional terminological knowledge, the system also will be able to retrieve the document from above as an answer to a query looking for visits of US government members to Europe.

So, in this approach, documents play a special role as a kind of context, and the IR system searches for contexts containing instances of the query concept. In contrast, with a single uniform knowledge base, documents are not treated in any special way, they are just concepts. This eases the representation of external attributes (which we do not consider here), but we are limited in expressing the content of documents. However, this problem can be overcome by combining terminological logic with Datalog (see section 7.3.5).

7.3 Datalog

7.3.1 Introduction

Datalog is a logic programming language that has been developed in the database field (see e.g. [Ullman 88], [Ceri et al. 90]). Like Prolog, it is based on horn logic. However, in contrast to Prolog, it does not allow for functions as terms, and the use of negation is restricted. Due to these constraints, there are sound and complete evaluation algorithms for Datalog — in contrast to Prolog, where certain programs cannot be evaluated. Now we first show how simple document retrieval can be formulated in

Datalog, and then we demonstrate the usage of Datalog for advanced IR applications like hypertext, aggregated documents and document type hierarchies.
For modelling simple document retrieval, we assume that there is an extensional predicate (a database relation) `docTerm(D,T)`, where each ground fact gives for a document `D` a term `T` the document is indexed with, e.g.:

```
docTerm(d1,ir). docTerm(d2,ir).
docTerm(d1,db). docTerm(d2,oop).
```

A query now can be formulated as a logical formula involving the predicate `docTerm`, e.g.

```
?- docTerm(D,ir).
?- docTerm(D,ir) & docTerm(D,db).
?- docTerm(D,ir) ; docTerm(D,db).
```

Here the first query searches for documents about IR, the second one for documents both about IR and DB, whereas the third one looks for documents dealing with IR or DB (the semicolon denotes disjunction here).
In order to allow for the more powerful inference mechanisms described in the following sections, a query should not relate directly to the extensional predicate `docTerm(D,T)`. Instead, we use a more general predicate `about(D,T)`, for which we will add new rules below. As basic rule, we define

```
about(D,T) :- docTerm(D,T).
```

Thus, the queries from above can also be formulated by replacing the predicate `docTerm` by `about`.

7.3.2 Hypertext structure

Hypertext structures contain links between single documents (or nodes). Often, there are different types of links with different semantics. For simplicity, we assume that there is a single type of link, although our approach can be extended easily to semantically richer structures. For representing links, we use a predicate `link(D1,D2)`, where a ground fact states that there is a directed link from `D1` to `D2`, e.g.:

```
link(d1,d2). link(d2,d3). link(d3,d1).
```

Now we assume that in retrieval, a document also deals with a certain topic if it refers to another document dealing with this topic. This can be written as

```
about(D,T) :- link(D,D1) & about(D1,T).
```

This rule makes `about` a recursive predicate. Thus if we want to prove `about(D,T)` for some document `D`, we look at the document itself as well as at those connected either directly or indirectly by links. Given the example link structure from above (see also figure 7.3), the cyclic link structure also raises the problem of possible cycles in the inference process; however, Datalog can cope with these cycles.
It should be emphasized that this way of considering hypertext links also allows for retrieval of nodes for which no index terms are given directly. For example, if we only

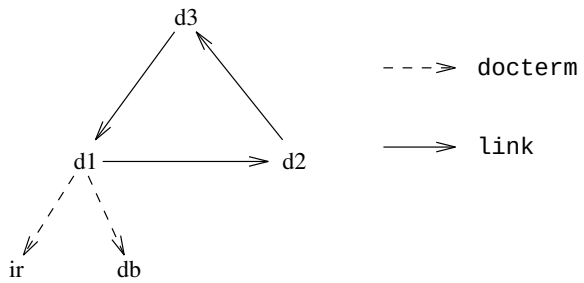


Abbildung 7.3: Hypertext retrieval

have an automatic indexing method for text, then nodes containing multimedia data but no text can be retrieved via their links to textual nodes.

7.3.3 Aggregation

As an example of documents structured hierarchically, assume that we have books consisting of several chapters, where each chapter in turn may contain several sections. Then the hierarchical structure can be represented by additional predicates, e.g.

```
book(b1).              section(b1c1,b1c1s1).
chapter(b1,b1c1).      section(b1c1,b1c1s2).
chapter(b1,b1c2).      section(b1c1,b1c2s1).
```

Here `chapter(B,C)` states that chapter `C` is a chapter of book `B`, and `section(C,S)` stands for the fact that `S` is a section in chapter `C`.

For performing retrieval, we abstract from the specific structural properties by introducing a predicate `part(D,P)` stating that `P` is a part of `D`. So we define the rules

```
part(D,P) :- chapter(D,P).
part(D,P) :- section(D,P).
```

Let us assume that index terms may be assigned (by means of the predicate `docTerm`) at arbitrary levels of the document structure hierarchy. In addition, we assume that a document or a part of it deal with a certain topic if a part of it is about this topic. Thus, we define an additional rule for the predicate `about`:

```
about(D,T) :- part(D,P) & about(P,T).
```

For retrieval, it may not be resonable to return a whole document (or a chapter) consisting of several parts if the topic asked for occurs only in a single part. In this case only the single part dealing with the topic from the query should be returned. So a more appropriate retrieval rule would be to return the document only if all its parts are about the current topic. In predicate logic, this can be formulated as

$$\begin{aligned}\forall D\,\forall T\ \mathrm{about}(D,T) \quad \Leftarrow \quad & \exists X\ \mathrm{part}(D,X) \wedge (\forall P\ \mathrm{part}(D,P) \Rightarrow \mathrm{about}(P,T)) \Leftrightarrow \\ & \exists X\ \mathrm{part}(D,X) \wedge \neg(\exists P\ \neg(\mathrm{part}(D,P) \Rightarrow \mathrm{about}(P,T))) \Leftrightarrow \\ & \exists X\ \mathrm{part}(D,X) \wedge \neg(\exists P\ \mathrm{part}(D,P) \wedge \neg(\mathrm{about}(P,T)))\end{aligned}$$

(In order to use a notation similar to Datalog, capital letters denote variables and predicate names start with lowercase letters here). The last formula can be written in Datalog with negation as follows:

```
about(D,T) :- part(D,X) & about(X,T) & not(abpart(D,T)).
abpart(D,T) :- part(D,P), not(about(P,T)).
```

Here we have added the subgoal `about(X,T)` to the first formula in order to make the rule safe, i.e. to avoid problems with negation in Datalog.

7.3.4 Object hierarchy

When we have different types of documents in a database, we can apply concepts from object-oriented databases for modelling the relationships between the different classes. As an example, consider the object classes shown in figure 9.13. Attributes marked with an asterisk denote set-valued attributes, and attributes in parentheses correspond to derived attributes. Bold arrows denote is-a relationships, i.e. the source is a subclass of the target, being inherited all attributes from its superclass, and the arrow from attribute `inbk` of class `inbook` to the class `book` illustrates the fact that instances of the latter class are values for this attribute.

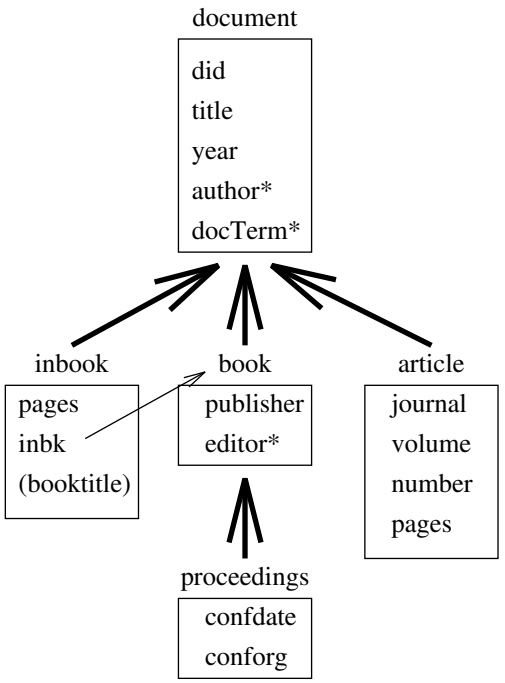


Abbildung 7.4: Example document class hierarchy

Some instances of these classes are represented by the following facts:

```
book(d1).
title(d1,''Introduction to IR'').
year(d1,1983).
author(d1,salton).
author(d1,mcgill).
docTerm(d1,ir).
publisher(d1,mcgraw).

proceedings(d2).          inbook(d3).
title(d2,''sigir94'').    inbk(d3,d2).
publisher(d2,springer).   pages(d3,100).
conforg(d2,acmsigir).     author(d3,cooper).
confdate(d2,jul94).
```

In addition, we need rules deriving attribute values not directly stored with an object, as for the class `inbook` here:

```
year(D,Y) :- inbk(D,B) & year(B,Y).
booktitle(D,T) :- inbk(D,B) & title(B,T).
```

In retrieval, when we search for a document, we should be able to retrieve also objects from the classes `book`, `article`, `inbook` and `proceedings`. Queries searching for `book`s also should be able to retrieve elements of its subclass `proceedings`. This can be accomplished by the following rules:

```
document(D) :- book(D).
document(D) :- inbook(D).
document(D) :- article(D).
book(D) :- proceedings(D).
```

Then a query for books or proceedings about IR can be formulated as

```
?- book(D) & about(D,ir).
```

7.3.5 Retrieval with terminological knowledge

If there is a thesaurus defining relationships between index terms, we can exploit these relationships for retrieval.

In case we only have a hierarchy of index terms, the hierarchical relation can be given by means of ground facts for the predicate `bt(NT,BT)`, stating that term `BT` subsumes (is a broader term than) term `NT`. Then we can add the following rule to the predicate `about`:

```
about(D,T) :- bt(T1,T) & about(D,T1).
```

Now assume that MIRTL is used for modelling the terminological knowledge. With certain restrictions, MIRTL definitions and connotations also can be transformed into Datalog programs (for the details, see [Fuhr 95]). For example, the following definitions

```
square        = (and quadrangle regular-polygon)
german-paper  = (and paper (c-some author german))
student-paper = (and paper (all author student))
multilingual  = (and person (atleast 2 speaks-lang))
```

can be mapped onto the Datalog rules

```
square(X)       :- quadrangle(X) & regular-polygon(X).
german-paper(X) :- paper(X) & author(X,Y) & german(Y).
student-paper(X):- paper(X) & not(non-student-author(X)).
multilingual(X) :- person(X) & speaks-lang(X,A) & speaks-lang(X,B) & A#B.
```

In addition, since we have defined concepts here, we can also perform inferences in the other direction, as for example with the first definition:

```
regular-polygon(X) :- square(X)
quadrangle(X)      :- square(X).
```

For performing retrieval based on MIRTL, we use the approach outlined in section 7.2.4.2, where documents are separate assertional modules (or worlds). This idea leads us to a different mapping of MIRTL constructs onto Datalog: As additional parameter, we always carry the document in which a concept or role is instantiated. So the example from above could be represented as follows:
`regular-triangle(D,X) :- triangle(D,X) & regular-polygon(D,X).`
However, this type of formulation poses difficulties when we try to combine it with the retrieval rules from the previous sections. For example, for hypertext links we would like to have a rule stating that a document also can be about a concept if the document refers to another one which contains this concept. Treating concepts as predicates would yield formulas in second order predicate calculus. We can circumvent this problem by using a single predicate `concept(C,D,I)`, where `D` refers to the document, `C` to the concept name and `I` to the concept instance. In a similar way, we use the predicate `role(R,D,F,T)` for a role `R` with arguments `F` and `T` (see below). This way, we would represent our example from above as follows:

```
concept(regular-triangle,D,X) :- concept(triangle,D,X) &
                                 concept(regular-polygon,D,X).
```

As an example involving roles, consider the definition of the concept father as
`father = (and (man (atleast 1 child)))`
which can now be expressed as
`concept(father,D,X) :- concept(man,D,X) & role(child,D,X,Y).`
With this approach, documents appear as explicit concepts only in case we talk about attributes of documents (in contrast to the content of documents as before). For this purpose, attributes of documents have to be mapped onto the corresponding roles, e.g.

```
role(author,D,D,A) :- author(D,A).
```

As an example of a concept referring explicitly to documents, consider the following rule for the concept `german-paper` mentioned before:

```
concept(german-paper,D,X) :- concept(paper,D,X) &
                             role(author,D,X,Y) & concept(german,D,Y).
```

It should be noted that a query searching for instances of this concept may not only retrieve documents which are `german-paper`s themselves. There also may be a (non-german) document talking about another document being a `german-paper`.

Terminological retrieval now works as follows: a user formulates a query concept which — in most cases — will not refer explicitly to any documents. As answer to a query, the system returns instances of this concept, each together with the document in which the instance occurred.

The retrieval rules from the previous sections have to be reformulated for terminological retrieval. This can be achieved by replacing the predicate `about`, generating one rule for the predicate `concept` and another one for `role`. For example, the hypertext rules from section 3 can be formulated as follows:

```
concept(C,D,I) :- link(D,D1) & concept(C,D1,I).
role(R,D,F,T) :- link(D,D1) & role(R,D1,F,T).
```

However, if there is a link between two documents containing contradictory information, then these rules will produce a contradiction in the knowledge base and thus leading to a failure. This problem could be overcome by using a special type of links between document pairs of this kind, for which the rules from above should not be applied.

7.4 Probabilistic Datalog

7.4.1 Introduction

In the logical approach to information retrieval (IR) as presented in [Rijsbergen 86], retrieval is interpreted as uncertain inference. For a query q, the system is searching for documents d which imply the query with a high probability $P(q \leftarrow d)$.

In section 7.3, we have describe the application of Datalog to IR. Since Datalog is based on certain inference, we are now describing a probabilistic version of Datalog called Datalog$_P$.

The major advantages of Datalog$_P$ are the following:

- The rule-based approach allows for easy formulation of retrieval models for specific or novel applications, like e.g. combination with a thesaurus or retrieval in hypertext bases or hierarchically structured documents.
- Classical probabilistic IR models can be formulated in Datalog$_P$ by appropriate rules, since they are just special cases of a more general probabilistic inference

mechanism.

- Since Datalog$_P$ allows for recursive rules, it provides more powerful inference than any other (implemented) probabilistic IR model.
- Finally, since Datalog$_P$ is a generalization of (deterministic) Datalog, it can be used as a standard query language for both IR and database systems, and thus also for integration of these two types of systems on the logical level.

7.4.2 Informal description of Datalog$_P$

Probabilistic Datalog is an extension of ordinary Datalog. On the syntactical level, the only difference is that with ground facts, also a probabilistic weight may be given, e.g.

```
0.7 indterm(d1,ir).  0.8 indterm(d1,db).
```

Informally speaking, the probabilistic weight gives the probability that the following predicate is true. In our example, document d1 is with probability 0.7 about IR and with probability 0.8 about databases (DB). Retrieving documents dealing with both of these topics now can be accomplished by means of the the rule

```
q1(X) :- indterm(X,ir) & indterm (X,db).
```

Obviously, document d1 fulfills predicate q1 with a certain probability. Let us assume that index terms are stochastically independent. Then we can compute a probability of 0.56 for the probabilistic AND-combination in this example. In a similar way, the OR-combination produced by the rules

```
q2(X) :- indterm(X,ir).
q2(X) :- indterm(X,db).
```

would give us probability 0.94 for `q2(d1)`.

As a more interesting example, we can use Datalog$_P$ rules for performing retrieval in hypertext structures where we have directed links between single documents (or nodes). Assume that these link also have probabilistic weights, e.g.

```
0.5 link(d2,d1).  0.4 link(d3,d2).
```

The idea behind these weights is the following: If we have a link from D1 to D2, and D2 is about a certain topic, then there is a certain probability that D1 is about the same topic. This probability is specified by the weight of the link predicate. Now we can formulate the rules

```
about(D,T) :- indterm(D,T).
about(D,T) :- link(D,D1) & about(D1,T).
```

Note that due to the recursive definition, a document also may be about a term if it is only indirectly linked to another document indexed with this term. Thus, the query

```
?- about(X,db).
```

now would return three documents, namely d1 with probability 0.8, d2 with probability $0.5 \cdot 0.8 = 0.4$ and d3 with probability $0.4 \cdot 0.5 \cdot 0.8 = 0.16$.

This example indicates that the idea of combining Datalog with probabilities yields very powerful retrieval methods. However, if we want to consequently apply probability theory, then we soon run into difficulties. Assume that in our hypertext structure, we search for documents both about IR and DB (similar to `q1`):

```
q4(X) :- about(X,ir) & about(X,db).
```

Then simple multiplication of the probabilistic weights involved in the inference process would give us for document d2: $0.5 \cdot 0.7 \cdot 0.5 \cdot 0.8 = 0.14$. This is not correct, since the probability for the link between d2 and d1 is considered twice; thus, the proper result would be 0.28. Besides counting the same probabilistic event twice, this simple approach also is unable to consider disjointness of complex events, for example when we search for documents either about IR or DB, but not about both:

```
q5(X) :- irnotdb(X).
q5(X) :- dbnotir(X).
irnotdb(X) :- indterm(X,ir) & not(indterm (X,db)).
dbnotir(X) :- indterm(X,db) & not(indterm (X,ir)).
```

If we would assume probabilistic independence of the subgoals of q5 (although they are disjoint events), we would compute the invalid result $1-(1-0.7 \cdot 0.2) \cdot (1-0.8 \cdot 0.3) \approx 0.35$ instead of the correct probability 0.38 for `q5(d1)`. The only way to overcome this problem in general is to switch from extensional semantics to intensional semantics (see e.g. [Pearl 88, pp. 4–12] for the comparison of these two approaches to uncertainty reasoning). For this purpose, we must keep track of the events that contribute to a derived fact.

In Datalog, there are two classes of predicates: For extensional database (EDB) predicates only ground facts, but no rules are given, wheras for intensional database (IDB) predicates, only rules are specified. In Datalog$_P$, we assume that each fact for an EDB predicate corresponds to a basic (probabilistic) event, and assign it an unique event key. A fact derived for an IDB predicate relates to a Boolean combination of basic events of the EDB facts from which this fact was derived. Thus, we assign IDB facts additionally an event expression consisting of a Boolean combination of the event keys of the corresponding EDB facts.

Throughout the examples given in the following, we will use the first letter of the EDB predicate along with the argument constants as event keys. For IDB facts, we will denote the event expression in brackets. Thus, we have, for example,

```
q1(d1) [i(d1,ir) & i(d1,db)]
q4(d2) [l(d2,d1) & i(d1,ir) & l(d2,d1)& i(d1,db)]
q5(d1) [i(d1,ir) & ¬ i(d1,db) | ¬ i(d1,ir) & i(d1,db)]
```

(where '`|`' denotes disjunction and '¬' negation). Given these Boolean expressions, we can identify identical events occurring more than once or disjoint events (e.g. the complement of an event). Then the corresponding probabilities can be computed correctly by means of the sieve formula.

In the following, we first describe syntax and semantics of Datalog$_P$. Then we present an approach for evaluating Datalog$_P$ programs. As with other probabilistic logics, the probability for a derived formula is mostly an interval and not a point value. For this reason, we consider a special variant of Datalog$_P$, where any two basic events are either independent or disjoint. In this case, always point values can be computed. Finally, we present further application examples of our approach.

7.4.3 Syntax

As basic elements, we have in Datalog *variables* (starting with capital letters), *constants* (numbers or alphanumeric strings starting with lower-case letters) and *predicates* (alphanumeric strings starting with lower-case letters).
A *term* is either a constant or a variable. Note that as a major difference to Prolog, Datalog does not allow for functions in terms. Thus, a *ground term* in Datalog can only be a constant, and the *Herbrand Universe* of a Datalog program is the set of constants occurring in it.
An *atom* $p(t_1, \ldots, t_n)$ consists of an n-ary predicate symbol p and a list of arguments $(t_1, \ldots, t_n)$ such that each t_i is a term. A *literal* is an atom $p(t_1, \ldots, t_n)$ or a negated atom $\neg p(t_1, \ldots, t_n)$.
A *clause* is a finite list of literals, and a *ground clause* is a clause which does not contain any variables. Clauses containing only negative literals are called *negative clauses*, while *positive clauses* are those with only positive literals in it. An *unit clause* is a clause with only one literal.
Horn clauses contain at most one positive literal. There are three possible types of Horn clauses, for which additional restrictions apply in Datalog:

1. *Facts* are positive unit clauses, which also have to be ground clauses.
2. *Rules* are clauses with exactly one positive literal. The positive literal is called the *head*, and the list of negative literals is called the *body* of the rule. In Datalog, rules also must be *safe*, i.e. all variables occurring in the head also must occur in the body of the rule.
3. A *goal clause* is a negative clause which represents a query to the Datalog program to be answered.

In Datalog, the set of predicates is partitioned into two disjoint sets, *EPred* and *IPred*. The elements of *EPred* denote extensionally defined predicates, i.e. predicates whose extensions are given by the facts of the Datalog program, while the elements of *IPred* denote intensionally defined predicates, where the extension is defined by means of the rules of the Datalog program. Furthermore, there are built-in predicates like e.g. $=, \neq, <$, which we do not discuss explicitly here.
If S is a set of positive unit clauses, then $E(S)$ denotes the extensional part of S, i.e. the set of all unit clauses in S whose predicates are elements of *EPred*. On the other

hand, $I(S) = S - E(S)$ denotes the intensional part of S (clauses in S with at least one predicate from *IPred*).
Now we can define a *Datalog program* P as a finite set of Horn clauses such that for all $C \in P$, either $C \in EDB$ or C is a safe rule where the predicate occurring in the head of C belongs to *IPred*.
So far, we have described the syntax of pure Datalog. In order to allow also for negation, we consider an extension called *stratified Datalog*. Here negated literals in rule bodies are allowed, but with the restriction that the program must be *stratified*. For checking this property, the *dependency graph* of a Datalog program P has to be constructed. For each rule in P, there is an arc from each predicate occurring in the rule body to the head predicate. P is stratified iff whenever there is a rule with head predicate p and a negated subgoal with predicate q, then there is no path in the dependency graph from p to q.
The syntax of Datalog$_P$ is only slightly different to that of stratified Datalog. A Datalog$_P$ program P consists of two sets P_E and P_I such that $P = P_E \cup P_I$. The intensional part P_I is a set of stratified Datalog rules, with the syntax of single rules as shown in the examples above. The extensional part P_E is a set of probabilistic ground facts of the form αg, where g is a ground fact and α is a probabilistic weight with $0 < \alpha \leq 1$. A probabilistic weight of 1 can be omitted. Furthermore, ground facts must be unique, i.e. $\alpha g \in P_E \wedge \alpha' g' \in P_E \wedge g = g'$ implies that $\alpha = \alpha'$.

7.4.4 Semantics of Datalog$_P$

For describing the semantics of Datalog$_P$, we use Herbrand semantics in combination with possible worlds.
The *Herbrand base (HB)* of a Datalog program is the set of all positive ground unit clauses than can be formed with the predicate symbols and the constants occurring in the program. Furthermore, let *EHB* denote the extensional and *IHB* the intensional part of *HB*. An *extensional database (EDB)* is a subset of *EHB*, i.e. a finite set of positive ground facts.
In deterministic Datalog, a Herbrand interpretation is a subset of the Herbrand base *HB*. For pure Datalog, there is a least Herbrand model such that any other Herbrand model is a superset of this model.
Stratified Datalog is based on a closed-world assumption. If we have rules with negation, then there is no least Herbrand model, but possibly several minimal Herbrand models, i.e. there exists no other Herbrand model which is a proper subset of a minimal model. Among the different minimal models, the one chosen is constructed in the following way: When evaluating a rule with one or more negative literals in the body, first the set of all answer-facts to the predicates which occur negatively in the rule body is computed (in case of EDB predicates these answer-facts are already given), followed by the computation of the answers to the head predicate. For stratified Da-

talog programs, this procedure yields an unique minimal model (for the details, see e.g. [Ullman 88, pp. 128–139] or [Ceri et al. 90, 211–228]). In the following, we will call the minimum model computed this way the *perfect Herbrand model*.

Now we turn to the possible world semantics of a Datalog$_P$ program P. In order to benefit from the work on the semantics of deterministic Datalog, we first consider only the set of probabilistic facts P_E. By removing the probabilistic weights of the ground facts, we can transform P_E into the corresponding set $\bar{P}_E$ of (deterministic) Datalog clauses, for which there exists a least Herbrand model $HL(\bar{P}_E)$. The interpretation of P_E is a set of possible worlds, where each world is a subset of $HL(\bar{P}_E)$.

In order to integrate also the intensional part P_I, we first have to go back from a single possible world to its corresponding deterministic Datalog program. Since for Datalog programs consisting of extensional clauses only, (i.e. relational databases) the mapping between such a program and its least Herbrand model is bijective, the set of extensional clauses $P(w_E)$ corresponding to a world w_E is uniquely defined. Then we can construct a deterministic Datalog program by combining these extensional clauses $P(w_E)$ with the original set of intensional clauses, i.e. $P_I \cup P(w_E)$. The corresponding possible world w of this program is its perfect Herbrand model $HP(P_I \cup P(w_E))$. One important property of this model is that it contains w_E as a subset.

Having described the structure of the possible worlds for a Datalog$_P$ program, we now turn to the interpretation of the probabilistic weights given with the ground facts. For that, let $M = (\mathcal{W}, \mu)$ denote a probability structure, where $\mathcal{W}$ is the set of possible worlds as described above and μ is a discrete probability distribution on $\mathcal{W}$. Furthermore, let v denote a valuation (a mapping from variables onto constants), g a ground fact and r a rule of the form $L_0 :- L_1 \& \ldots \& L_n$.

Then we can recursively define the notion of an *extension* $[t]_{(M,w,v)}$ of a probabilistic term t for a valuation v and a world w of M and the notion of the *truth* $(M, w, v) \models \phi$ of a formula ϕ for a valuation v and a world w of M by means of the following rules:

1. $[z]_{(M,w,v)} = z$
2. $[g]_{(M,w,v)} = \mu(\{w' \in \mathcal{W} : (M, w', v) \models g\})$
3. $(M, w, v) \models \alpha g$ iff $[\alpha]_{(M,w,v)} = [g]_{(M,w,v)}$
4. $(M, w, v) \models F$ iff F is a positive fact and $F \in w$, or F is a negative fact and $|F| \notin w$
5. $(M, w, v) \models L_0 :- L_1 \& \ldots \& L_n$ iff whenever $\{L_1 v\} \in w \wedge \ldots \wedge \{L_n v\} \in w$ then $\{L_0 v\} \in w$

Extensions are defined by the first two rules. Rule 1 states that the extension of a rational constant is always the rational number it represents. The second rule specifies that the extension of a fact is computed as the sum of the probabilities of the worlds in which the fact is true. In rule 3, the truth of probabilistic facts of the form αg is defined by stating that the extension of the fact g must equal the extension of the probabilistic weight α. The last two rules define the truth of facts and rules in the

same way as in deterministic Datalog; here $|F|$ denotes the positive counterpart in case F is a negative fact.
With these concepts, we can finally define the validity of a derived (probabilistic) fact. A *theory of Datalog$_P$* is a set Φ of formulae which is closed under logical consequence; i.e. Φ is such that, if $\phi \in \Phi$ and ϕ' is true in all worlds in which ϕ is true, then also $\phi' \in \Phi$. A formula ϕ is *valid in a theory* Φ of Datalog$_P$, written $\models_\Phi \ \phi$, iff ϕ is true in all worlds w in which all formulae in Φ are also true.
Datalog$_P$ as described so far is a special case of the probabilistic $\mathcal{L}_2$ logic as presented in [Halpern 90] and [Nilsson 86]. It defines precise probabilities only for the ground facts given by the Datalog$_P$ program. As a simple exmple, the program

```
0.9 docTerm(d1,ir).
```

has the interpretation
$P(W_1) = 0.9$: {docTerm(d1,ir)}
$P(W_2) = 0.1$: {}
For derived facts, the semantics as described above only yields certain constraints on the probabilities. For example, the program

```
0.4 docTerm(d1,ir).   0.5 docTerm(d1,db).
q1(X) :- indterm(X,ir) & indterm (X,db).
```

has — among others — the following interpretations:

I_1:
$P(W_1) = 0.2$: {docTerm(d1,ir)}
$P(W_2) = 0.2$: {docTerm(d1,ir), docTerm(d1,db)}
$P(W_3) = 0.3$: {docTerm(d1,db)}
$P(W_4) = 0.3$: {}

I_2:
$P(W_1) = 0.5$: {}
$P(W_2) = 0.4$: {docTerm(d1,ir), docTerm(d1,db)}
$P(W_3) = 0.1$: {docTerm(d1,db)}

I_3:
$P(W_1) = 0.4$: {docTerm(d1,ir)}
$P(W_2) = 0.5$: {docTerm(d1,db)}
$P(W_3) = 0.1$: {}
Thus, we get for the probability of the answer:
$0.0 \leq P(\texttt{docTerm(d1, ir)}\&\texttt{docTerm(d1, db)}) \leq 0.4$,
depending on the overlap between the worlds in which `indterm(d1,ir)` is true and those in which `indterm(d1,db)` is true. This outcome reflects a careful interpretation of the knowledge that we have in this case. One possibility for getting more specific probabilities for derived facts is by entering also probabilities for Boolean combinations of probabilistic facts, e.g.

```
0.3 indterm(d1,ir) & indterm(d1,db).
```

This approach is taken by the probabilistic logics mentioned above. However, instead of extending the syntax of Datalog$_P$ in order to be able to specify additional probabilities, we prefer to follow another approach which is usually taken in IR: due to the fact that it is impracticable or impossible to estimate the probabilities of arbitrary combinations of single events, one uses to make additional assumptions about the independence of events. In our example, this would mean that we prefer interpretation I_1, thus yielding the result
$P(\texttt{docTerm(d1, ir)}\&\texttt{docTerm(d1, db)}) = 0.2$.
The general procedure is described in more detail below.

7.4.5 Evaluation of Datalog$_P$ programs

Our approach of evaluating Datalog$_P$ programs is based on the notion of *event keys* and *event expressions*. As an *event*, we regard any probabilistic fact, whether given extensionally or derived via rules. Let us assume that there is a set **EK** of event keys which is a set of unique identifiers plus two special elements $\top$ and $\bot$ for events (facts) which are always true or false, respectively. Now we define a mapping $\varepsilon : \bar{P}_E \rightarrow \mathbf{EK} - \{\bot\}$ which assigns each ground fact g an event key $\varepsilon(g)$, with the following constraints:

1. $\forall g\,(1g) \in P_E \Leftrightarrow \varepsilon(g) = \top$
2. $\forall g, g'\,\varepsilon(g) = \varepsilon(g') \wedge g \neq g' \Rightarrow (1g) \in P_E \wedge (1g') \in P_E$

The first rule states that probabilistic facts with weight 1 are assigned the event key $\top$, and the second rule requests that two different ground facts have the same event key only in case they have probability 1.
Given this mapping, we can recursively define event expressions $\eta(F)$ for any derived or extensional fact F:

1. $\eta(F) := \bigvee_{v:F=(Lv)}\ \eta(Lv)$
2. $\eta(Lv) := \bigvee_r\ \eta(rv)$, where r is a rule whose head matches (Lv)
3. $\eta(L_0 :- L_1\& \ldots \& L_n\ v) := \eta(L_1 v) \wedge \ldots \wedge \eta(L_n v)$
4. $\eta(Lv) := \neg\eta(|L|v)$, if L is a negative literal.
5. $\eta(g) = \varepsilon(g)$, if $g \in \bar{P}_E$.

Here the first rule states that we form the disjunction over all valuations v that make this fact true. In a similar way, rule 2 specifies that all rules whose head matches the current goal (Lv) have to be applied, and the final event expression is formed as the disjunction of the event expressions resulting from these rules. The third rule describes the forming of event expressions for Datalog$_P$ rules, and rule 4 states that the event expression of negative goals is formed as the negation of the event expression of the corresponding positive goal. Finally, the event expression of a ground fact is its event key.
Let **EE** denote the set of all event expressions that can be formed this way. Then **EE** together with the Boolean operators forms a Boolean algebra.

Given the set **EE** and an interpretation of a Datalog$_P$ program, we can specify a mapping $\omega : \mathbf{EE} \rightarrow \mathcal{P}(\mathcal{W})$ from event expressions onto sets of possible worlds, namely the set of worlds that make the fact corresponding to the event expression true: $\omega(\eta(F)) = \{w|F \in w\}$.
By means of the following rules, we can define an isomorphism between the Boolean algebra on **EE** and set algebra on $\mathcal{W}$ (see also [Rölleke 94]):

1. $\omega(\eta(g)) = \{w|g \in w\}$, if g is a ground fact.
2. $\omega(\eta(F_1) \wedge \eta(F_2)) = \omega(\eta(F_1)) \cap \omega(\eta(F_2))$
3. $\omega(\eta(F_1) \vee \eta(F_2)) = \omega(\eta(F_1)) \cup \omega(\eta(F_2))$
4. $\omega(\neg\eta(F)) = \mathcal{W} - \omega(\eta(|F|))$.

Thus we have shown that for any fact F derived by a Datalog$_P$ program, the corresponding event expression $\eta(F)$ gives us via the mapping $\omega(\eta(F))$ the set of worlds in which F is true.
In principle, the probability of F being true can be computed as $\mu(\{w|w \in \omega(\eta(F))\})$. In our approach, we compute the probability based on the event expression, by exploiting the fact that we have a Boolean algebra on **EE**, and that we can apply the axioms from this algebra in order to transform event expressions. In general, the probability of an event expression $e \in \mathbf{EE}$ can be computed according to the formula

$$P(e) = \mu(\omega(e)).$$

In the case of complex event expressions, the well-known sieve formula can be applied in order to compute this probability. For that, we first have to transform the event expression into disjunctive normal form (DNF), that is:

$$e = K_1 \ \vee \ldots \vee \ K_n,$$

where the K_i are event atoms or conjunctions of event atoms, and an event atom is either an event key or a negated event key (n is the number of conjuncts of the DNF). From Boolean algebra, we know that any Boolean expression can be transformed into DNF. Now we can apply the sieve formula:

$$\begin{aligned} P(e) &= P(K_1 \ \vee \ldots \vee \ K_n) \\ &= \sum_{i=1}^{n} (-1)^{i-1} \left(\sum_{\substack{1 \leq j_1 < \\ \ldots < j_i \leq n}} P(K_{j_1} \ \wedge \ldots \wedge \ K_{j_i}) \right). \end{aligned} \tag{7.1}$$

For computing the probabilities for the AND-combination of conjuncts, we can simplify the corresponding expressions by means of the axioms of Boolean algebra. Combinations containing an event atom and its negation yield $\perp$ and thus can be ignored. Furthermore, duplicate event keys in a combination can be removed. This way, the

probability of an event expression is finally computed as the sum of probabilities of conjunctions of event atoms. For example, the event expression for `q5(d1)` in section 2 leads to the following computation:

$P($`i(d1,ir) & ¬ i(d1,db) | ¬ i(d1,ir) & i(d1,db)`$) =$
$P($`i(d1,ir) & ¬ i(d1,db)`$) + P($`¬ i(d1,ir) & i(d1,db)`$) -$
$P($`i(d1,ir) & ¬ i(d1,db) & ¬ i(d1,ir) & i(d1,db)`$) =$
$P($`i(d1,ir) & ¬ i(d1,db)`$) + P($`¬ i(d1,ir) & i(d1,db)`$)$

As discussed before, there are now two possibilities for going on:

1. As in probabilistic logics, we chose a careful interpretation of the probabilistic information available. Then we can compute a point value for the probability of an event expression only if the probabilities of all conjuncts formed in the sieve formula are given explicitly. Otherwise, only a probability interval can be given as a result.
2. We use additional assumptions about the independence of events, thus allowing us to compute always a point value.

Here we follow the second approach, which is described in more detail in the following section.

7.4.6 Datalog$_P$ with independence assumptions

As mentioned already in the introduction, the major goal of our approach is the development of a probabilistic inference scheme for IR which is not only more powerful than classic probabilistic IR models, but also can be implemented efficently. For this reason, we consider only a special case of Datalog$_P$, where we assume that events in general are independent. Furthermore, there may be sets of events which are disjoint to each other. This version of probabilistic Datalog is called Datalog$_{PID}$ (probabilistic Datalog with independence and disjointness).

Independence of events means that the probability of the conjunction of two events equals the product of the probability. That is, for any two independent events with keys e_1, e_2,

$$P(e_1 \wedge e_2) = P(e_1) \cdot P(e_2).$$

This assumption is suitable for most IR applications. With respect to eqn 7.1, this means that we can compute the probability of a conjunct of event atoms as the product of the probabilities of the single event atoms. If the event atom is an event key, then we take the probability given with the corresponding probabilistic ground fact, and in the case of a negated event key, the complement probability is to be taken. Thus, we get for the event expression from our last example:

$P($`i(d1,ir) & ¬ i(d1,db)`$) + P($`¬ i(d1,ir) & i(d1,db)`$) =$
$P($`i(d1,ir)`$) \cdot (1 - P($`i(d1,db)`$)) + (1 - P($`i(d1,ir)`$)) \cdot P($`i(d1,db)`$).$

In addition to independent events, we also consider the case of disjoint events. As an example, assume that we have only uncertain information about the publication years of books, e.g.

```
0.2 py(d3,89). 0.7 py(d3,90). 0.1 py(d3,91).
```

Here the publication year of d3 is either 89, 90 or 91, with the probabilities 0.2, 0.7 and 0.1, respectively. Obviously, the facts relating to one document represent disjoint events. In databases, this situation is called "imprecise attribute values". Thus, the interpretation of this program would be:
$P(W_1) = 0.2$: {`py(d3,89)`}
$P(W_2) = 0.7$: {`py(d3,90)`}
$P(W_3) = 0.1$: {`py(d3,91)`}
Now a query for books published after 89

```
?- py(X,Y) & Y > 89.
```

would yield the event expression `[p(d3,90) | p(d3,91)]` for d3. In terms of the sieve formula, we have
$P(\texttt{p(d3,90)}|\texttt{p(d3,91)}) = P(\texttt{p(d3,90)}) + P(\texttt{p(d3,91)}) - P(\texttt{p(d3,90)}\&\texttt{p(d3,91)})$.
When computing the probability for this expression, we must consider that the two events here are disjoint, thus the probability of the conjunction is 0, and we get the correct result (as can be sen from the interpretation) as the sum of the two event probabilities $0.7 + 0.1 = 0.8$.
In general, if two events with keys e_1, e_2 are disjoint, this means that there is no possible world in which the corresponding facts both are true (i.e. $\omega(e_1 \wedge e_2) = \emptyset$), and thus we have

$$P(e_1 \wedge e_2) = 0.$$

7.4.7 Further application examples

In addition to the examples presented in the previous sections, here we want to give a few more examples which illustrate the strength of our approach.
So far, we have considered only Boolean combinations of terms as IR queries. However, we can also express probabilistic query term weighting in Datalog$_{PID}$. This is accomplished by regarding the terms in a query as disjoint events, as in the following program:

```
0.8 docTerm(d1,db).    0.7 docTerm(d1,ir).
0.4 qtw(db).           0.6 qtw(ir).

s(D) :- qtw(X) & docTerm(D,X).
?- s(D).
```

The interpretation of this program depicted in figure 7.5 shows that the worlds of `qtw(db)` and `qtw(ir)` are disjoint, whereas `docTerm(d1,db)` and `docTerm(d1,ir)`

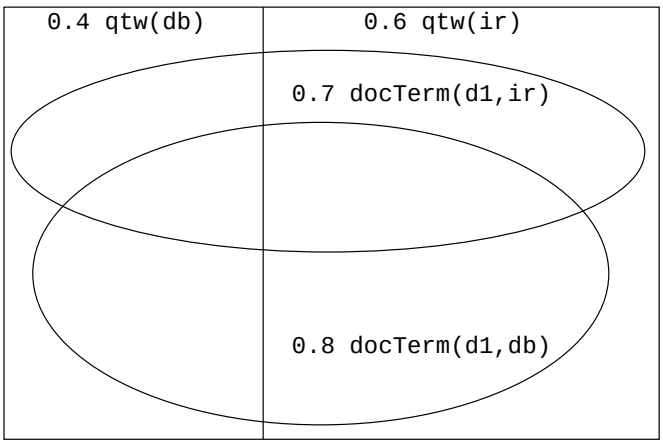


Abbildung 7.5: Interpretation of query term weighting example

are independent events. As event expression of the result, we get
`d1 [q(db) & dT(d1,db) | q(ir) & dT(d1,ir)]`.
Since `q(db)` and `q(ir)` are disjoint, the result is computed like a scalar product of query and document, namely $0.4 \cdot 0.8 + 0.6 \cdot 0.7 = 0.74$. Thus, we can achieve the same weighting scheme as for example in the vector model or in INQUERY ([Turtle & Croft 91]).
As another extension of probabilistic Datalog, vague predicates can be used. For example, assume that a PC shop offers the following models (tuples denote model name, CPU type, memory size, dsik size and price:

```
pc(m1,486/dx50,8,540,900).
pc(m2,pe60,16,250,1000).
pc(m3,pe90,16,540,1100).
```

Now a customer asks for a model with a price less than 1000:
`?- pc(MOD, CPU, MEM, DISK, PRICE), PRICE < 1000`
Obviously, it would not be appropriate to interpret this condition in a Boolean way. Rather, there is a certain probability that the customer finally will pay more than 1000 in case he gets a special bargain. This situation can be modelled by interpreting the condition `PRICE < 1000` probabilistically by means of a vague predicate $\hat{<}$, that would yield the following results:

```
1.00 ^<(1000,900)
1.00 ^<(1000,950)
0.99 ^<(1000,1000)
0.90 ^<(1000,1050)
0.60 ^<(1000,1100)
```

Thus, formulating the query using this predicate
`?- pc(MOD, CPU, MEM, DISK, PRICE), PRICE ^< 1000`

should yield the outcome

```
1.00 pc(m1,486/dx50,8,540,900).
0.99 pc(m2,pe60,16,250,1000).
0.60 pc(m3,pe90,16,540,1100).
```

Internally, the application of vague predicates generates new event keys which are considered in the final probability computation. Thus, for the last answer, the event expression would be

`pc(m3) & ` $\hat{<}$(1000,1100).

Vague predicates are essential for advanced IR applications. Besides vague fact conditions (as in this example), it can be used for proper name search or retrieval of OCRed text (based on string similarity). Most important, the notion of similarity used in most multimedia IR approaches (e.g. audio retrieval or image retrieval) can be interpreted as vague predicates.

7.4.8 Probabilistic rules

So far, probabilistic weights have only been assigned to ground facts. In many cases, one also would like to assign weights to rules, as for example

```
0.7 likes-sports(X) :- man(X).
0.4 likes-sports(X) :- woman(X).
man(peter).
```

stating that 70% of all men like sports, but only 40% of all women. Given only the information that Peter is a man, we would like to infer a probability of 0.7 that he likes sports. Thus, the interpretation of this program should be as follows:

$P(W_1) = 0.7$: {`man(peter), likes-sports(peter)`}
$P(W_2) = 0.3$: {`man(peter)`}

In terms of event expressions, the query

```
?- likes-sports(peter)
```

would generate the expression

`l-s_1(peter) & man(peter)`,, where `l-s_1(peter)` is an event generated by the first rule, with a probability of 0.7.

As a similra example, consider the case where also the sex of a person is unknown:

```
0.7 l-s(X)        :- sex(X,male).
0.4 l-s(X)        :- sex(X,female).
0.5 sex(X,male)   :- human(X).
0.5 sex(X,female) :- human(X).
human(peter).
```

Give the additional information that `sex(X,male)` and `sex(X,female)` are disjoint for any `X`, the interpretation of this program would be:

$P(W_1) = 0.35$: {`sex(peter,male), l-s(peter)`}
$P(W_2) = 0.15$: {`sex(peter,male)`}

$P(W_3) = 0.20$: {sex(peter,female), l-s(peter)}
$P(W_4) = 0.30$: {sex(peter,female)}
Asking wheter or not peter likes sports would yield the event expression
(sex_1(peter,male) & l-s_1(peter) | sex_2(peter,female) & l-s_2(peter))
& human(peter)
from which we can derive the correct probability $0.5 \cdot 0.7 + 0.5 \cdot 0.4 = 0.55$.
However, rules must be formualted in a cautious way, as will be seen from the following example. Assume that documents are linked either by a reference link or through the fact that they share the same author. Then we could formulate the program

```
sameauthor(D1,D2) :- author(D1,X) & author(D2,X).
0.5 link(D1,D2) :- refer(D1,D2).
0.2 link(D1,D2) :- sameauthor(D1,D2).
```

However, what happens if there is both a reference link and the same author?

```
?? link(D1,D2) :- refer(D1,D2) & sameauthor(D1,D2).
```

For this case, no probability is defined, nor can it be computed (by means of some kind of independence assumption), since we do not have enough parameters specified. More generally speaking, given the conditional probabilities $P(l|r)$ and $P(l|s)$, we have no information about $P(l|r \wedge s)$.
In order to avoid this type of situation, rules must be specified in such a way that (for deterministic facts on the right and side), at most one rule for a predicate can match. Thus, in the last example, we could formulate the rules

```
0.7 link(D1,D2) :- refer(D1,D2) & sameauthor(D1,D2).
0.5 link(D1,D2) :- refer(D1,D2) & not(sameauthor(D1,D2)).
0.2 link(D1,D2) :- sameauthor(D1,D2) & not(refer(D1,D2)).
```

In fact, this situation is the same as with probabilistic inference networks, where we have to specify a complete link matrix for the combination of probabilistic events (see figure 7.6). In probabilistic Datalog, the link matrix is specified by means of a set of rules

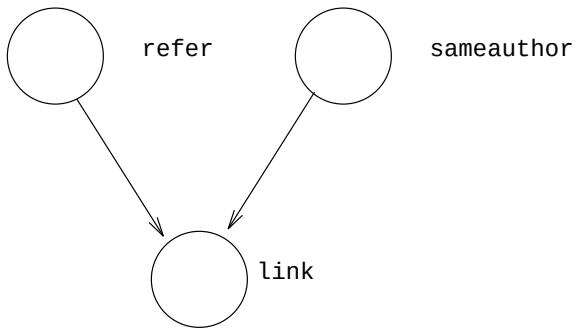


Abbildung 7.6: Example inference network

Kapitel 8

Implementierung von IR-Systemen

Nachdem in den vorangegangenen Kapiteln zunächst Verfahren zur Repräsentation von Textinhalten und eine Reihe von IR-Modellen vorgestellt wurden, soll nun in diesem Kapitel die Implementierung dieser Ansätze beschrieben werden. Hierzu gehen wir zunächst kurz auf die dafür relevanten Hardware-Aspekte ein und skizzieren den globalen Aufbau von IRS. Den Schwerpunkt des Kapitels bilden die Darstellungen von standardisierten Dokumentarchitekturen (ODA und SGML) und von Zugriffspfadstrukturen, die in IRS eingesetzt werden (Scanning, invertierte Listen, Signaturen, PAT-Bäume).

8.1 Hardware-Aspekte

In diesem Abschnitt sollen nur einige Kennzahlen von heute verfügbarer Rechnerhardware aufgelistet werden. Wir beschränken uns dabei auf den Bereich der Workstations — zum einen, weil Großrechner nur noch für kommerzielle Online-IRS eingesetzt werden, zum anderen, weil mittlerweile auch PCs in den Leistungsbereich von Workstations stoßen, so daß eine Unterscheidung zwischen diesen beiden Rechnerklassen überflüssig ist. Die Rechenleistung von Workstations liegt heute typischerweise im Bereich zwischen 50 und 300 Mips (Millionen Instruktionen pro Sekunde) mit Hauptspeicherausbauten ab 32 MB.

8.1.1 Speichermedien

8.1.1.1 Optische Speichermedien

Optische Speichermedien sind für IRS relevant, weil sie hohe Speicherkapazitäten bei niedrigen Kosten bieten. Zur Zeit sind drei verschiedene Typen von optischen Platten auf dem Markt:

Typ	Kapazität	Latency	Positio-nierzeit	Trans-ferrate	Kosten Gerät	Kosten Datenträger
	MB	ms	ms	MB/s	TDM	DM/MB
Magnetplatten	200-8000	2.3-4.6	7-12	3-20		0.4
ROD	128-1300	12	30-40	0.6	9	1
WORM	650			0.8	1	0.16
CD-ROM	650			1.2	0.2	0.02
DVD	4700-17000			1.1		
Disketten	0.7-3				0.2	2
Magnetkassetten	200-5000				1	0.3
DAT	500-5000				1.5	0.01
Video-8	5000				2.5	0.01

Tabelle 8.1: Kennzahlen heutiger Speichermedien

- **CD-ROM** besitzen eine Kapazität von 600 MB und bei n-Speed-Laufwerken eine Transferrate von n * 170 kB/sec. Sie erlauben nur lesenden Zugriff. Der Herstellungsaufwand zur Produktion einer Master-CD ist relativ hoch, die Stückkosten jedoch relativ niedrig. Dadurch eignen sich CDs in erster Linie für die Verteilung von Datenbasen in ausreichender Stückzahl. Zahlreiche kommerzielle Datenbasis-Produzenten bieten daher neben dem online-Zugriff auf Hosts ihre Datenbasen auch in der Form von CDs an, für die allerdings jährlich Updates nachgekauft werden müssen.
  Eine Weiterentwicklungen hiervon ist die **Video-Disk**, die kurz vor der Markteinführung steht. Die DVD kann ein- oder zweiseitig jeweils in einer oder in zwei Lagen beschrieben werdenen, woraus sich eine Kapazität zwischen 4.7 und 17 GB ergibt. Die Übertragungsrate wird 1.1 MB/s betragen.
- **WORM**-Platten (Write Once Read Many) sind nur einmal beschreibbar und nicht löschbar. Dadurch sind sie in erster Linie für die dauerhafte Archivierung geeignet. Es gibt sie mit Kapazitäten von 650 MB bis 9,6 GB.
- **ROD** (Rewritable Optical Disk) sind dagegen beliebig oft beschreibbar und eignen sich daher auch für temporäre Daten. Ihre Kapazität liegt zwischen 128 MB und 2 GB.

Für alle Arten von optischen Platten gibt es sogenannte Jukeboxen, die für eine einzelne Aufnahme-/Wiedergabestation mit Hilfe eines Wechslers mehrere Platten aufnehmen können.

8.1.1.2 Kennzahlen von Speichermedien

Tabelle 8.1 zeigt einige Kennzahlen heutiger bei Workstations eingesetzter Speichermedien. Diese Zahlen müssen in Relation gesetzt werden zum Speicherplatzbedarf

von Dokumenten: Man rechnet etwa 2 KB für eine Schreibmaschinenseite. In Literaturdatenbanken benötigt man im allgemeinen 2–4 KB für die Kurzfassung eines Dokumentes zusammen mit den bibliographischen Angaben. Eine durchschnittliche Textdatei (z.B. in einem Bürosystem) belegt etwa 10 KB, eine Graphik 20 KB. Ein Text-Faksimile mit 300 dpi benötigt in komprimierter Form 50 KB, ein Bild-Faksimile in gleicher Auflösung komprimiert 200 KB. Für Video im MPEG-Format werden je nach Auflösung 1.5 bis 4 Mbit/s benötigt.

8.1.2 Ein-/Ausgabegeräte

Zur maschinellen Erfassung von Dokumenten werden heute bereits in weitem Maße **Scanner** eingesetzt. Diese erreichen Auflösungen bis zu 600 dpi ("dots per inch", also ≈ 0.04 mm). Kombiniert man diese Geräte mit geeigneter Zeichenerkennungssoftware, so erhält man praktisch einsetzbare **Klarschriftleser**. Allerdings muß wegen der nicht zu vernachlässigenden Fehlerquote eine manuelle Nachbearbeitung der Texte erfolgen. Die zweite wichtige Anwendung von Scannern ist das Einlesen von Graphiken und von Dokumenten in Faksimile-Darstellung.
Für die Ausgabe von Dokumenten steht ein breites Spektrum von Druckern zur Verfügung. Weit verbreitet sind **Laserdrucker** (sowohl schwarzweiß als auch in Farbe), die Auflösungen bis zu 1200 dpi erreichen. Höhere Auflösungen bietet noch der Fotosatz mit bis zu 2000 dpi.
Gebräuchliche **Monitore** mit 1–2 Millionen Pixels bieten dagegen nur Auflösungen im Bereich 75–100 dpi. Aus diesem Grund werden für CAD und Desktop Publishing teilweise Geräte mit höherer Auflösung eingesetzt.

8.1.3 Kommunikationsnetzwerke

Durch den Trend zu verteilten Informationssystemen gewinnt die Leistungsfähigkeit der verwendeten Kommunikationsnetzwerke an Bedeutung. Weit verbreitet ist z.B. die **Client-Server-Architektur**, bei der die Datenbestände auf einem Fileserver gehalten werden und erst bei Bedarf zu der Workstation, die die Dokumente benötigt, übertragen werden. Heute sind folgende Kommunikationsnetzwerke im praktischen Einsatz:

- **Ethernet** verwendet Koaxialkabel, twisted pair oder FDDI (fiber distributed data interface) als Trägermedium und erreicht Übertragungsraten bis zu **100 Mbit/sec**.
- **ATM**-Netze erreichen Übertragungsraten bis in den Bereich von **1 GB/s**. Im Vergleich zu Ethernet sind aber hier die Anschlußkosten deutlich höher.
- **ISDN** (integrated standard digital network) ist der von den öffentlichen Telefongesellschaften angebotene Standard zur Datenübermittlung. Aufgrund der Beschränkung auf herkömmliche Telefonleitungen werden hier maximal **64 Kbit/sec**

pro Datenkanal übertragen. Ein einzelner Anschluß bietet dabei zwei Datenkanäle und einen zusätzlichen Steuerkanal.

8.2 Aufbau von IRS

8.2.1 Funktionale Sicht

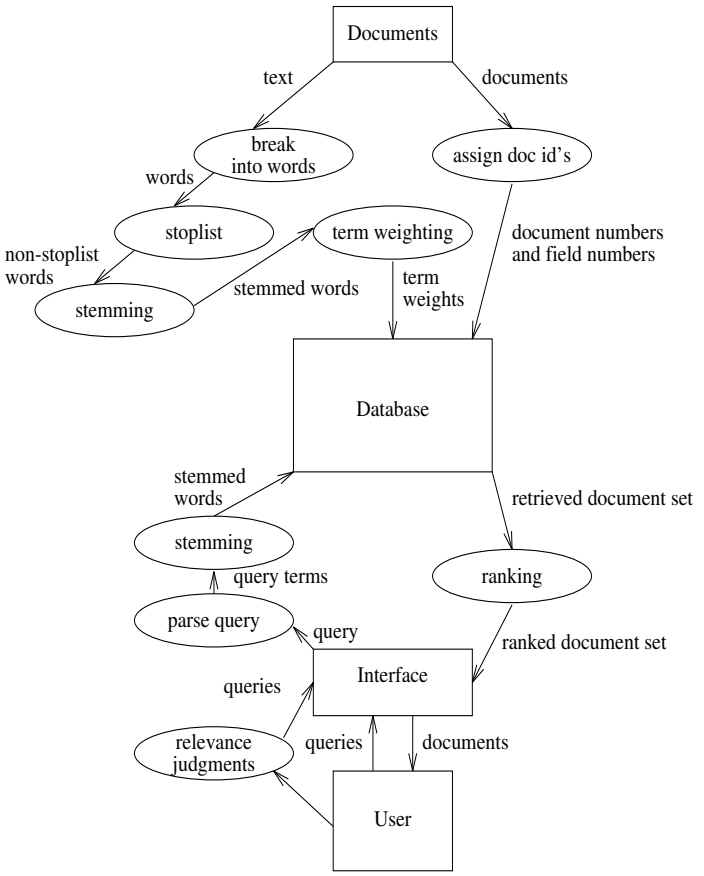


Abbildung 8.1: Funktionale Sicht eines IRS

Abbildung 8.1 (aus [Frakes & Baeza-Yates 92]) zeigt die funktionale Sicht auf ein IRS.

8.2.2 Dateistruktur

Abbildung 8.2 zeigt die Dateistruktur eines typischen IRS. Neben der **Dokument-Datei** mit den eigentlichen Dokumenten sind meist noch zwei zusätzliche Dateien

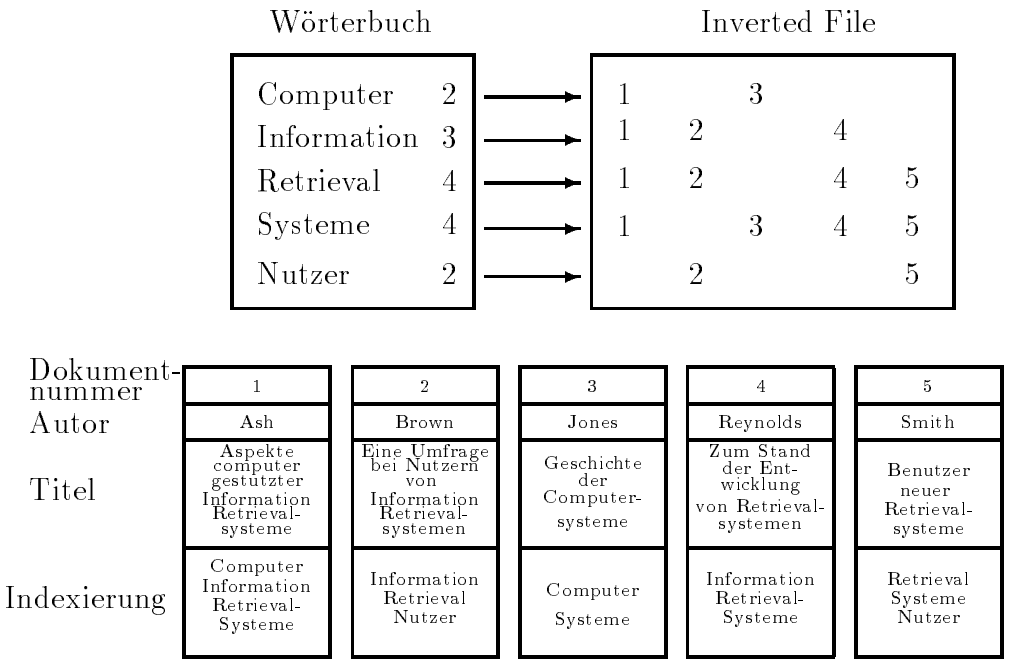


Abbildung 8.2: Dateistruktur eines IRS

vorhanden: Das **Wörterbuch** enthält die in den Dokumenten vorkommenden Terme zusammen mit ihrer Dokumenthäufigkeit; dadurch kann ein Benutzer sich bei der Formulierung einer Anfrage über die Verteilung der in Frage kommenden Terme informieren. Zur Beschleunigung der Suche ist hier als spezieller Zugriffspfad ein **inverted file** skizziert. Dieses enthält für jeden Term die Liste derjenigen Dokumentnummern, in denen der Term vorkommt. Ein IRS muß dafür sorgen, daß bei Änderungen in der Dokument-Datei die beiden Hilfsdateien jeweils aktualisiert werden.
Meist ist die Dateistruktur eines IRS noch wesentlich komplexer, z.B. wenn ein Thesaurus unterstützt werden soll.

8.2.3 Dialogfunktionen herkömmlicher IRS

Ein herkömmliches IRS bietet für Dialogretrieval meist folgende Funktionen an:

- Zugangskontrolle (Datenschutz),
- Auswahl der Datenbasis (meist werden mehrere Datenbasen mit einem einzigen IRS verwaltet),
- Anzeige des Wörterbuchs / Thesaurus',
- Formulierung von Anfragen,
- Anzeige von Antworten (einschließlich Drucken und Downloading),

- Verwaltung von Suchprofilen (zur späteren Modifikation und für periodisch wiederkehrende Retrievalläufe mit den neu hinzugekommenen Dokumenten (SDI, selective dissemination of information))

8.3 Dokumentarchitekturen

Um Dokumente in einem IRS zu speichern, muß ein Eingabe- und Darstellungsformat für diese festgelegt werden. Frühere IRS gingen meist von einfachen ASCII-Texten aus, doch heute stellt man höhere Ansprüche an die Gestaltung von Dokumenten. In technisch-wissenschaftlichen Datenbasen möchte man zumindest Formeln und nach Möglichkeit auch Graphiken ablegen können. Bei Büroinformationssystemen sollen die mit gebräuchlichen Textverarbeitungssystemen erstellten Dokumente beim Retrieval in ihrem Originalformat angezeigt werden, und außerdem soll die Möglichkeit bestehen, die Dokumente mit dem Textverarbeitungssystem dann weiter zu bearbeiten. Neben der Dokumentenerstellung spielen die Dokumentformate auch eine Rolle beim elektronischen Dokumentenaustausch: im einfachsten Fall soll das Dokument beim Empfänger nur originalgetreu ausgedruckt werden, häufig möchte der Empfänger aber auch das Dokument weiter bearbeiten (im wissenschaftlichen Bereich werden z.B. häufig Dokumente im TeX-Format per E-mail übertragen). Da aber eine Vielzahl von Textverarbeitungs bzw. Desktop-Publishing-Systemen mit jeweils unterschiedlichen Dokumentformaten auf dem Markt sind, müssen Standards definiert werden, die den Austausch von Dokumenten zwischen den verschiedenen Systemen erlauben. Insbesondere müssen auch IRS entsprechende Eingabe- und Ausgabeschnittstellen beinhalten (z.B. kann man mit einigen kommerziellen IRS Dokumente im SGML-Format verwalten).
In den folgenden Abschnitten stellen wir die zwei wichtigsten standardisierten Dokumentformate vor: ODA ist eine primär europäische Entwicklung für den Bürobereich, während SGML aus den USA stammt und zunächst im technisch-wissenschaftlichen Bereich Verbreitung fand.

8.3.1 ODA

ODA (**Office Document Architecture**) definiert ein Dokumentarchitektur-Modell, das die Be- und Verarbeitung von Dokumenten durch unterschiedliche Systeme ermöglicht. Das zugehörige Austauschformat zur Übertragung von Dokumenten zwischen verschiedenen Systemen heißt ODIF (**Office Document Interchange Format**). Die nachfolgende Beschreibung orientiert sich im wesentlichen an der Darstellung in [Krönert 88].

8.3.1.1 Grundkonzepte von ODA

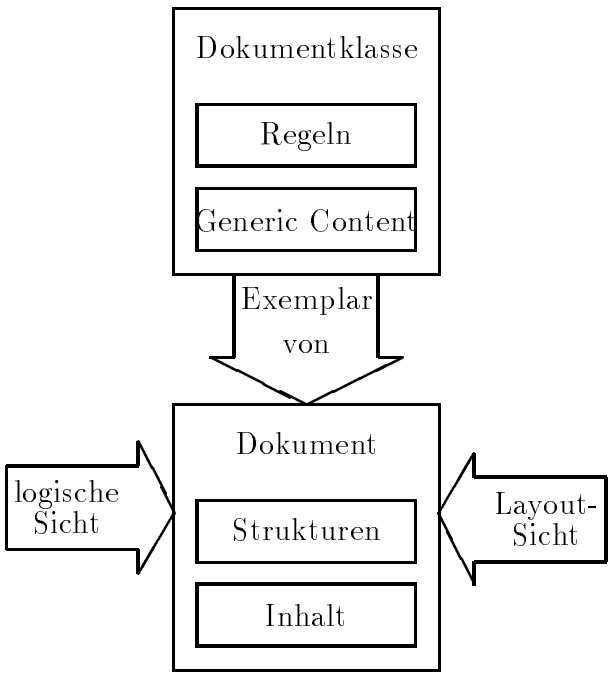


Abbildung 8.3: Dokumente aus der Sicht von ODA

Das wesentliche neue Konzept von ODA gegenüber früheren Dokumentformaten ist die zweifache Strukturierung des Inhalts eines Dokumentes:

1. Die **logische Struktur** beinhaltet die Unterteilung eines Dokumentes in seine logischen Einheiten, also Kapitel, Abschnitte, Sätze, Bilder, Tabellen usw.
2. Die **Layout-Struktur** umfaßt dagegen die Darstellungsaspekte in Form einer Unterteilung des Dokumentes in Seiten und diese wiederum rechteckige Bereiche.

Jede der beiden Strukturen wird im wesentlichen als Hierarchie von Objekten dargestellt; zusätzlich gibt es einige nicht-hierarchische Relationen (z.B. für Verweise auf Fußnoten).
Der eigentliche Inhalt[1] eines Dokumentes ist auf die Objekte der untersten Stufe verteilt (nur diese können Inhalt besitzen). Dabei sind verschiedene **Inhaltsarchitekturen** möglich. Im Standard sind bislang Text, Rastergraphik und Vektorgraphik erfaßt. Die Norm ist allerdings durch Hinzufügen weiterer Inhaltsarchitekturen leicht erweiterbar.

[1] hier nicht im Sinne der inhaltlichen Bedeutung eines Textes

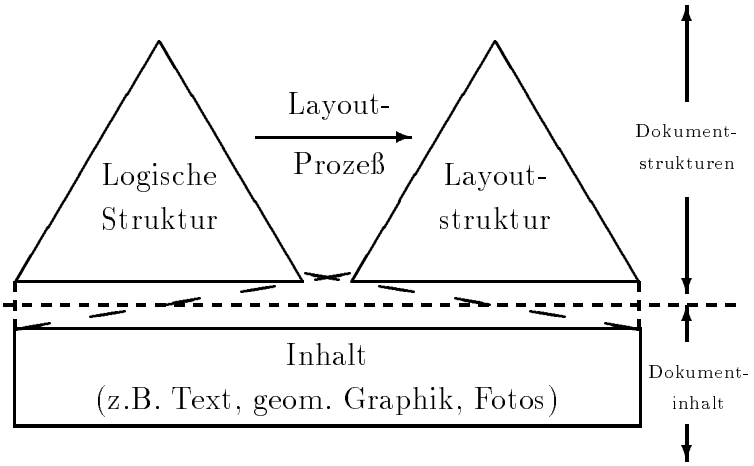


Abbildung 8.4: Dokumentstrukturen und Dokumentinhalt

Jedes logische und Layout-Objekt ist ein Exemplar einer Objektklasse, die vorher definiert sein muß. Insbesondere gehört auch ein Dokument als Objekt zu einer Dokumentklasse genannten Objektklasse. Die Definition einer Dokumentklasse besteht aus der Definition der verwendeten Objektklassen; zusätzlich können hier vorgegebene Inhaltsstücke (generic content) für Objekte bestimmter Objektklassen definiert werden, wie z.B. Logos oder Standardparagraphen.

8.3.1.2 Strukturen in ODA

Sowohl logische wie auch Layout-Struktur werden als Hierarchie von Objekten dargestellt. Dabei gehört jedes Objekt zu einer **Objektklasse** und einem **Objekttyp**. Objekttypen sind in der Norm definiert zusammen mit den darauf anwendbaren Attributen und ihrer Rolle in der Dokumentarchitektur. Objektklassen werden in der Dokumentklassendefinition auf der Basis von Objekttypen spezifiziert.
Für die logische Struktur sind in ODA folgende Objekttypen definiert:

- **document logical root** als oberste Stufe der logischen Struktur,
- **basic logical object** als unterste Ebene der logischen Struktur (Blätter des Strukturbaumes), die die Inhaltsportionen des Dokumentes enthalten, und
- **composite logical object** auf den Hierarchieebenen zwischen document logical root und basic logical object.

Für die Layout-Struktur sind folgende Objekttypen vorgegeben:

- **document layout root** als oberste Stufe der Layout-Struktur,
- **page set** als Zusammenfassung einer Gruppe von Seiten,
- **page** als zweidimensionaler Bereich, auf dem der Dokumentinhalt positioniert und dargestellt wird,

- **frame** als rechteckiger Bereich auf einer Seite, in den der Inhalt bei der Layoutgestaltung formatiert werden kann. Ein frame enthält i.a. mehrere Blöcke, aber kein Block (außer er gehört zum generic content) darf außerhalb eines frames auftreten.), und
- **block**, der den Inhalt einer einzigen Inhaltsarchitektur enthält.

Die Zuordnung zwischen basic logical objects und blocks sieht nun so aus, daß sowohl mehrere basic logical objects einem block zugeordnet sein können (z.B. eine Kapitelüberschrift bestehend aus Nummer und Text) als auch umgekehrt mehrere blocks zu einem basic logical object (z.B. Aufteilung eines Absatzes auf zwei Seiten).
Attribute beschreiben die Eigenschaften von Objekten. Jedes Attribut hat einen Attributtyp und einen Wert. Der Attributtyp legt die Semantik eines Attributs fest (z.B. Dimension für Objekte vom Typ page, frame und block, Position für Objekte vom Typ frame oder Block). Attribute sind entweder explizit bei Objekten angegeben oder aus **styles** abgeleitet. Styles sind Attributsammlungen bei der zugehörigen Objektklassenbeschreibung oder der hierarchisch übergeordneten Objektklasse.

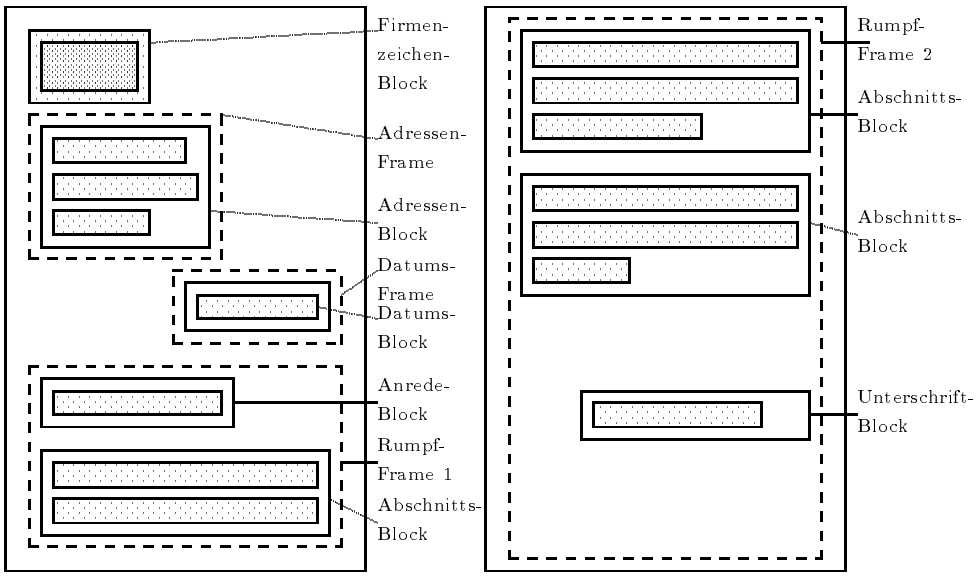


Abbildung 8.5: Aufteilung eines Dokumentes in Blöcke und Frames

8.3.1.3 Austauschformat

ODIF definiert die Darstellung eines ODA-Dokumentes als Bitstrom zum Zwecke der Übertragung zwischen verschiedenen Systemen. Hierbei gibt es verschieden mächtige Austauschformate. Diese variieren bzgl. folgender Merkmale:

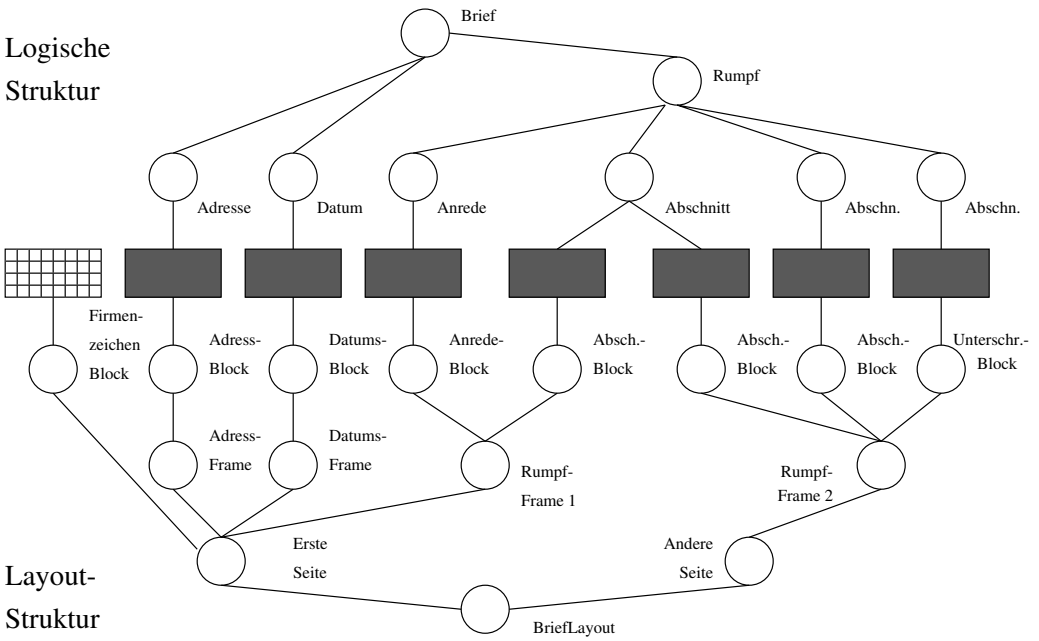


Abbildung 8.6: Logische und Layout-Sicht desselben Dokumentes

- der Menge der verwendbaren Inhaltsarchitekturen: Die character content architecture (Text) ist in allen Austauschformaten enthalten, komplexere Austauschformate enthalten zusätzliche Inhaltsarchitekturen.
- der Übertragung in weiterbearbeitbarer und/oder formatierter Form: Abhängig vom Zweck des Austauschs werden folgende Austauschformate unterschieden:
  - Die **formatierte Form** erlaubt nur das originalgetreue Reproduzieren des Dokumentes (z.B. Fax, Telex),
  - Die **weiterbearbeitbare Form** ermöglicht die Weiterbearbeitung des Dokumentes beim Empfänger, allerdings muß hierbei das Layout vom Empfänger definiert werden.
  - Bei der **weiterbearbeitbaren formatierten Form** wird das Dokument vollständig übertragen. Dadurch wird sowohl eine Weiterbearbeitung als auch eine originalgetreue Reproduktion des Dokumentes möglich.

8.3.1.4 SGML

SGML steht für „Standard Generalized Markup Language“. Es wurde ursprünglich im US-Verlagswesen als Format für die Ablieferung von Manuskripten in elektronischer Form definiert. Mittlerweile hat sich SGML aber zum weltweit führenden Standard für Dokumentformate entwickelt.

8.3.1.4.1 Markup-Ansätze

Markup-Sprachen dienen dazu, durch zusätzlich in den Text eingestreute Tags die automatische Verarbeitung von Texten zu steuern. Abhängig davon, was markiert wird, und wie die weitere Verarbeitung gesteuert wird, kann man verschiedene Arten von Markup unterscheiden:

1. Zeichensetzung: Markieren von Satzzeichen, um z.B. Satzendepunkte von Abkürzungspunkten unterscheiden zu können (in LaTeX .␣ vs. .\␣)
2. Layout: z.B. Markieren von Zeilen- und Seitenumbruch (in LaTeX`\linebreak`, `\pagebreak`)
3. prozedural: prozedurale Befehle zur Steuerung der Verarbeitung (wie z.B. in Troff, TeX, LaTeX)
4. deskriptiv: deskriptive Befehle zur Steuerung der Verarbeitung (wie z.B. in SGML)
5. referentiell: Verweise auf andere Dokumentteile, um diese entweder zu referenzieren oder aus anderen Dateien einzubinden (embed, include in SGML; `\ref`, `\include` in LaTeX)
6. Meta-Markup: Definition von Syntax und Semantik der eigentlichen Markup-Sprache (wie DTD (document type definition) in SGML, s.u.).

8.3.1.4.2 SGML-Standards

SGML (Standard Generalized Markup Language) ist durch den ISO-Standard Nr. 8879 definiert. Da SGML nur zur Definition der logischen Struktur von Dokumenten dient, aber (im Gegensatz etwa zu ODA) keinerlei Möglichkeiten zur Steuerung des Layouts vorsieht, gibt es ergänzend hierzu den DSSSL-Standard (Document Style Semantics & Specifications, ISO 10179), der eine Layout-Spezifikationssprache für SGML-Dokumente definiert. ODA ist im ISO-Standard 8613 normiert, und die logische Struktur von ODA-Dokumenten kann in ODML beschrieben werden, einer Sammlung von SGML-DTDs.

8.3.1.4.3 Eigenschaften von SGML

SGML einfach als Markup-Sprache zu bezeichnen, ist eine völlig unzureichende Beschreibung der Möglichkeiten dieser Sprache:

- Aufgrund der Ausdrucksstärke kann SGML auch als Datenbank-Definitions-Sprache dienen (z.B. um einen Produktkatalog zu verwalten). Es existieren bereits Software-Werkzeuge, die das Navigieren und Suchen in solchen Strukturen realisieren.
- SGML ist eine erweiterbare Dokument-Beschreibungssprache, da jederzeit neue Tags hinzugefügt werden können und dies in der Regel eine aufwärtskompatible Erweiterung ermöglicht.
- Als Metasprache dient SGML zur Definition von Dokumenttypen.

SGML unterstützt

- logische Dokumentstrukturen, insbesondere Hierarchien beliebiger Schachtelungstiefe,
- die Verknüpfung und Adressierung von Dateien, um z.B. ein Dokument auf mehrere Dateien aufzuteilen (Bilder werden stets in vom Text getrennten Dateien gehalten),
- Hypertext durch referentielle Verweise,
- Multimedia durch Einbinden nicht-textueller Darstellungen aus separaten Dateien.

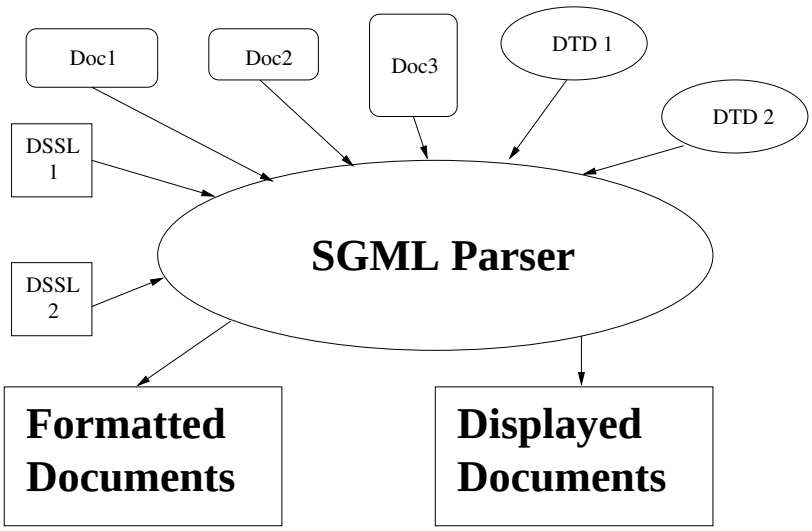


Abbildung 8.7: Verarbeitung von SGML-Dokumenten

8.3.1.4.4 Verarbeitung von SGML-Dokumenten

Abbildung 8.7 zeigt die möglichen Verarbeitungsschritte von SGML-Dokumenten. Nachdem ein SGML-Dokument entweder mit einem normalen Texteditor oder einem syntaxgesteuerten Editor (z.B. sgmls in Emacs) erstellt wurde, überprüft ein Parser die Syntax gemäß der zugehörigen DTD. Danach kann das Dokument für eine Kontext-orientierte Suche indexiert werden (einige kommerzielle IRS unterstützen bereits SGML-Dokumente) oder in eine andere Repräsentation übersetzt werden (zahlreiche Textsysteme ermöglichen den Im- und Export im SGML-Format). Um ein SGML-Dokument dastellen zu können, benötigt man zusätzlich eine DSSSL-Spezifikation (sowie entsprechende Software); für unterschiedliche Ausgabemedien kann es verschiedene solcher Spezifikationen geben.

8.3.1.4.5 SGML-Markup

SGML unterstützt vier Arten von Markup:

1. deskriptiv durch Tags,
2. referentiel durch Referenzen auf Objekte,
3. Meta-Markup durch Markup-Deklarationen in der DTD, und
4. prozedural durch explizites Einbinden weiterer DTDs mittels der Befehle LINK und CONCUR.

Am wichtigsten ist der deskriptive Markup. Ein einzelnes Element wie z.B.
`<SECTITLE ID=''SGML1'' AUTHOR=''Fuhr''> SGML-Demo</SECTITLE>`
besteht dabei aus folgenden Teilen:

- Der GI (generic identifier) kennzeichnet die Art des Elements in Start- und Ende-Tags (hier `<SECTITLE>` am Beginn und `</SECTITLE> am Ende`
- `ID=idref` (im Start-Tag) ordnet dem Element einen eindeutigen Identifier zu
- Attribut-Werte-Paare (im Start-Tag) dienen zur Zuweisung von Attributen
- Der eigentliche Inhalt steht zwischen Start- und Ende Tag

Alle diese Teile können — abhängig von der DTD — optional sein. Die Elemente haben ein Inhalts-Modell in Form einer Grammatik-Produktion, enthalten also selbst wieder andere Elemente. Das oberste Element ist jeweils das Dokument.

```
<!ELEMENT article - - (front, body) >
<!ELEMENT front O O title author+>
<!ELEMENT title - - #PCDATA>
<!ELEMENT author - - #PCDATA>
<!ELEMENT body O O section+>
<!ELEMENT section - O sectitle par*>
<!ELEMENT sectitle - - #PCDATA>
<!ELEMENT par - O #PCDATA>
<!ATTLIST article
            crdate CDATA #REQUIRED
            writer CDATA #REQUIRED>
<!ATTLIST body
           numbering CDATA #IMPLIED
           headings CDATA #IMPLIED>
```

Abbildung 8.8: Beispiel einer DTD

Abbildung 8.8 zeigt eine Beispiel-DTD und Abbildung 8.9 ein zugehöriges Dokument. In der DTD beginnt die Deklaration der einzelnen Elemente mit `<!ELEMENT`, gefolgt vom Elementnamen. Die nächsten beiden Argumente geben an, ob Start-und Ende-Tag optional (O) oder verpflichtend (-) sind. Danach wird die Grammatikproduktion des Elemntes in Form eines regulären Ausdrucks notiert. `#PCDATA` steht dabei für

```
<article crdate=150596 writer=fuhr>
<title> SGML-Beispiel </title>
<author> N. Fuhr </author>
<section> <sectitle>  1. Abschnitt </sectitle>
<par> 1.Paragraph
<par> 2.Paragraph
<section> <sectitle>  2. Abschnitt </sectitle>
</article>
```

Abbildung 8.9: Beispiel-Dokument zur Beispiel-DTD

beliebigen Text (als Inhalt eines Elements. Die Attributlisten der einzelnen Elemente werden gesondert beginnend mit `<!ATTLIST` deklariert, wobei für jedes Attribut dessen Typ anzugeben ist und ob das Attribut fehlen darf.
DTDs besitzen somit folgende Eigenschaften:

- Sie definieren eine Klasse von Dokumenten (analog zur Objektklasse in ODA).
- Sie spezialisieren SGML für Dokumente einer Klasse
- Sie beinhalten eine Attribut-Grammatik
- Sie beinhalten eine Schachtelungs-Grammatik
- Sie unterstützen Hierarchien durch Schachtelung

Als etwas komplexeres Beispiel einer DTD gibt Abbildung 8.10 einen Teil der HTML-DTD wieder. Hier taucht noch das syntaktische Element `<!ENTITY` auf, mit dem Makros definiert werden können; dadurch wird die zugehörige Grammatikproduktion überall dort eingesetzt, wo der Makroname auftaucht. Kommentare sind in doppelte Bindestriche eingeschlossen.

8.3.1.4.6 HTML

Durch die zunehmende Verbreitung des WWW ist HTML (Hypertext Markup Language) sehr populär geworden. Nur Web-Dokumente, die in diesem Format vorliegen, können die volle Hypermedia-Funktionaliat der Web-Browser ausnutzen. Dokumente in anderen Formaten müssen mittels externer Viewer angezeigt werden und können daher keine Links enthalten.
Betrachtet man HTML aus Sicht von SGML, so läßt sich folgendes feststellen:

1. HTML entspricht einer SGML-Dokumentklasse (DTD). Dadurch ist HTML weniger flexibel als SGML.
2. HTML enthält sowohl logische als auch Layout-Tags. Dadurch wird die strikte Trennung zwischen logischer Strukur und Layout aufgegeben. Zudem sind die logischen Tags auf ein Minimum beschränkt (i.w. Titel, Paragraph, Aufzählungen), was keine vernünftige logische Strukturierung ermöglicht. Dies erweist sich

```
<!ELEMENT HTML O O HEAD, BODY --HTML document-->
<!ELEMENT HEAD O O TITLE>
<!ELEMENT TITLE - - #PCDATA>
<!ELEMENT BODY O O %content>
<!ENTITY % content
      "(%heading | %htext | %block | HR)*">
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!ENTITY % htext "A | %text"    --hypertext-->
<!ENTITY % text "#PCDATA | IMG | BR">
<!ELEMENT IMG    - O EMPTY --Embed. image-->
<!ELEMENT BR    - O EMPTY>
<!ENTITY % block "P | PRE">
<!ELEMENT P     - O (%htext)+  --paragraph-->
<!ELEMENT PRE - - (%pre.content)+ --preform.-->
<!ENTITY % pre.content "#PCDATA | A">
<!ELEMENT A     - - (%text)+  --anchor-->
<!ELEMENT HR    - O EMPTY -- horizontal rule -->
<!ATTLIST A
        NAME CDATA #IMPLIED
        HREF CDATA #IMPLIED   --link--
        >
<!ATTLIST IMG
        SRC CDATA #REQUIRED     --URL of img--
        ALT CDATA #REQUIRED
        ALIGN (top|middle|bottom) #IMPLIED
        ISMAP (ISMAP) #IMPLIED
        >
```

Abbildung 8.10: DTD zu HTML (Auszug)

als nachteilig, wenn umfangreiche Dokumente in HTML-Form angeboten werden sollen oder wenn man mittels einer der Internet-Suchmaschinen Retrieval im WWW machen will.

3. Das Präsentationsformat für HTML ist mehr oder weniger verbindlich festgelegt. Da es aber nur wenige Layout-Tags gibt und zudem keine Möglichkeit zum Übermitteln von DSSSL-Spezifikationen besteht, sind die Formatierungsmöglichkeiten sehr begrenzt. Zahlreiche Anbieter versuchen dem dadurch zu begegnen, daß sie große Teile des Dokumentes als inline-Bilder darstellen (analog zur Darstellung von Formeln als inline-Graphik, was durch dasFehlen entsprechender Tags zur Auszeichnung von Formeln bedingt ist).

8.4 Zugriffspfade

8.4.1 Scanning

8.4.1.1 Generelle Überlegungen

Eine mögliche Alternative bei der Wahl eines geeigneten Zugriffspfades für ein IRS ist der Verzicht auf einen gesonderten Zugriffspfad. Stattdessen wird eine möglichst effiziente sequentielle Suche implementiert. Der Vorteil dieser Lösung ist die Einsparung des Overheads (Speicherplatz, Aktualisierungsaufwand) für einen Zugriffspfad. Andererseits hat diese Lösung eine Reihe von Nachteilen:

- Der Aufwand beim Retrieval (und damit die Antwortzeiten) wächst linear mit dem Datenvolumen. Daher ist Scanning nur für kleinere Datenbestände geeignet. Insbesondere wird Scanning auch in Texteditoren eingesetzt.
- Selbst wenn man nur den informatischen Ansatz zur Repräsentation von Textinhalten zugrundelegt, müssen truncation, Maskierung und Kontextoperatoren in den Scanning-Algorithmus integriert werden, was diesen außerordentlich komplex macht. Will man aber den computerlinguistischen Ansatz verwenden, dann wird die Komplexität nochmals deutlich erhöht; eine Alternative hierzu wäre die zusätzliche Abspeicherung der Texte in normierter Form, um in diesen anstelle der Originaltexte zu suchen — was aber wiederum eine deutliche Erhöhung des Speicherplatzbedarfs zur Folge hat.
- Problematisch ist auch die Kombination von Scanning mit Rankingalgorithmen. Da die inverse Dokumenthäufigkeit eines Terms erst nach dem Durchlaufen aller Dokumente feststeht, müßten alle Vorkommen aller Frageterme als Resultat des Scannings zwischengespeichert werden, bevor der Rankingalgorithmus angewendet werden kann

Wenn somit mehrere Gründe gegen den generellen Einsatz von Scanning in IRS sprechen, so gibt es dennoch einige wichtige Einsatzbereiche hierfür:

- Es gibt hardwaremäßige Implementierungen von Scanning-Algorithmen. Dabei sind diese Boards direkt an den zu den Platten führenden I/O-Kanal angeschlossen, und die Verarbeitungsgeschwindigkeit ist so hoch wie die Transferrate des Plattenlaufwerks. Dadurch werden die Dokumente mit maximaler Geschwindigkeit durchsucht. Durch die Verteilung der Dokumente auf mehrere Platten und die Zuordnung jeder Platte zu einem eigenen Board kann ein paralleles Arbeiten mit linearer Steigerung der Verarbeitungsgeschwindigkeit realisiert werden.
- Eine wichtige Dialogfunktion ist das Highlighting von Suchbegriffen bei der Anzeige von gefundenen Dokumenten. Das Lokalisieren der Suchbegriffe im Dokumenttext ist nur durch Scanning möglich.
- Werden Signaturen als Zugriffspfad eingesetzt, so liefern die Signaturen nur eine Obermenge der korrekten Antwortdokumente. Daher muß eine zusätzliche Vergleichskomponente implementiert werden, die die gültigen Dokumente in dieser Obermenge bestimmt. Hierfür kommen nur Scanning-Algorithmen in Frage.

8.4.1.2 Vorbemerkungen zu Scanning-Algorithmen

Die hier gewählte Darstellung von Scanning-Algorithmen orientiert sich im wesentlichen an der Beschreibung [Baeza-Yates 89]. Dort finden sich auch Literaturhinweise zu den einzelnen Algorithmen. Wir betrachten hier nur Patterns, die aus einer festen Zeichenfolge bestehen. Zu den meisten der vorgestellten Algorithmen existieren allerdings auch Varianten, bei denen die Patterns auch Alternativen oder maskierte Zeichen enthalten dürfen.
Einige Notationen werden im folgenden ständig benutzt:

n Länge des zu durchsuchenden Textes,
m Länge des Patterns (sei stets $m \leq$n),
c Größe des zugrundeliegenden Alphabets Σ,
$\bar{C}_n$ Erwartungswert für die Anzahl der zeichenweisen Vergleiche in einem Algorithmus für einen Text der Länge n.

Bei der Analyse des Aufwands gehen wir von einer zufälligen Zeichenkette[2] aus; dabei wird angenommen, daß eine Zeichenkette der Länge l aus der Konkatenation von l Zeichen besteht, die unabhängig und gleichverteilt zufällig aus Σ entnommen werden. Dann ist die Wahrscheinlichkeit für die Gleichheit von zwei zufällig ausgewählten Zeichen $1/c$ und die Wahrscheinlichkeit für die Übereinstimmung zweier Zeichenfolgen der Länge m ergibt sich zu $1/c^m$. Der Erwartungswert $E(t)$ der Anzahl Treffer t für ein Pattern der Länge m in einem Text der Länge n ist dann:

$$E(t) = \frac{n - m + 1}{c^m}. \tag{8.1}$$

[2]Da diese Annahme in der Realität kaum zutrifft, werden außerdem experimentelle Resultate mit englischem Text präsentiert.

8.4.1.3 Der naive Algorithmus

Beim naiven Algorithmus wird an jeder Position im Text mit dem ersten Zeichen des Patterns beginnend verglichen, bis entweder eine Ungleichheit auftritt oder das Ende des Patterns erreicht wird. Das nachfolgende Beispiel illustriert diese Vorgehensweise für das Pattern `abracadabra`; dabei ist beim Pattern jeweils das letzte Zeichen aufgeführt, mit dem verglichen wird.

```
aababcabcdabracadabra

ab
 abr
  a
   abr
    a
     a
      abr
       a
        a
         a
          abracadabra
```

Eine einfache Implementierung dieses Algorithmus' in C sieht wie folgt aus:

```
naivesearch(text, n, pat, m)                          /* Search pat[1..m] */
      char text[], pat[];
      int n, m;
{
  int i, j, k, lim;

  lim = n-m+1
    for(i = 1; i ≤ lim; i++) {                          /* Search */
      k = i;
      for(j = 1; j ≤ m && text[k] == pat[j]; j++)
        k++;
      if(j > m ) Report_match_at_position( i);
    }
}
```

Für Abschätzung des Aufwands beim naiven Algorithmus werden folgende Formeln angegeben:

- Erwartungswert für die Anzahl Vergleiche bis zum ersten Treffer:

$$\bar{C}_{firstmatch} = \frac{c^{m+1}}{c-1} - \frac{c}{c-1} \tag{8.2}$$

- Erwartungswert für die Gesamtzahl der Vergleiche:

$$\bar{C}_n = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right)(n - m + 1) + O(1) \tag{8.3}$$

(im worst case $m \cdot n$ Vergleiche).

Die oben gezeigte Implementierung des naiven Algorithmus' läßt sich deutlich verbessern, wenn die zugrundeliegende Rechnerarchitektur spezielle Maschinenbefehle zur Suche nach dem ersten Auftreten eines bestimmten Zeichens (bzw. eines Zeichens aus einer Menge von Zeichen) bereitstellt. Zum Beispiel gibt es in allen auf der IBM/360-Architektur basierenden Rechnern den Befehl "Translate and Test", der genau dies leistet. Dieser Maschinenbefehl kann in obigem Algorithmus für die Suche nach dem ersten Zeichen des Patterns eingesetzt werden. Noch effizienter ist allerdings die Suche nach dem Zeichen aus dem Pattern mit der geringsten Auftretenswahrscheinlichkeit.

8.4.1.4 Der Knuth-Morris-Pratt-Algorithmus

Dieser Algorithmus basiert auf folgender Idee zur Beschleunigung des naiven Algorithmus: Wenn bereits eine teilweise Übereinstimmung zwischen Pattern und Text gefunden wurde, bevor das erste verschiedene Zeichen auftritt, kann diese Information zur Wahl eines besseren Aufsetzpunktes gewählt werden. Dadurch wird vermieden, daß der Pattern beginnend mit jeder Textposition verglichen werden muß. Folgendes Beispiel illustriert diese Vorgehensweise für das Pattern `abracadabra`:

```
aababrabrabracadabra

ab
 abr
   abrac
       brac
          bracadabra
```

Das Beispiel illustriert zum einen die Verringerung der Anzahl Aufsetzpunkte im Vergleich zum naiven Algorithmus. Außerdem erkennt man aber auch, daß der Zeiger im Text nie zurückgesetzt werden muß.

Der Algorithmus erfordert eine Vorprozessierung des Patterns zur Erstellung einer Tabelle `next[1..m]`. Dabei gibt `next[i]` jeweils die nächste Position im Pattern an, mit der bei Ungleichheit an der Position `i` verglichen werden muß. Die Definition dieser Tabelle ist wie folgt:

$$\begin{aligned} next[j] &= \max\{i | (pattern[k] = pattern[j - i + k] \quad \text{für } k = 1, \ldots, i-1) \\ &\quad \wedge pattern[i] \neq pattern[j]\} \end{aligned}$$

oder

$$next[j] = \max(\{0\} \cup M[j]) \qquad \text{mit}$$

$$M[j] = \{i | 1 \leq i < j \wedge pattern[i] \neq pattern[j] \wedge \forall k\ 1 \leq k < i\ (pattern[k] = pattern[j - i + k])\}$$

Es wird also nach dem längsten übereinstimmenden Präfix gesucht, so daß das nächste Zeichen im Pattern verschieden ist von dem Zeichen, bei dem die Ungleichheit auftrat.
Beispiel: Für den Pattern `abracadabra` *ergibt sich folgende Tabelle* `next` *:*

```
          a b r a c a d a b r a
next[j]   0 1 1 0 2 0 2 0 1 1 0 5
```

Zusätzlich bedeutet `next[i]`=0, daß der Zeiger im Text um eins vorgerückt wird und wieder mit dem Anfang des Patterns vergleichen wird; `next[m+1]` definiert den Aufsetzpunkt im Fall eines Treffers.
Der in C implementierte Algorithmus sieht wie folgt aus:

```
kmpsearch( text, n, pat, m)              /* Search pat[1..m] in text[1..n] */
      char text[], pat[];
      int n, m;
{
  int j, k, resume, matches;
  int next[MAX_PATTERN_SIZE];

  pat[m+1] = CHARACTER_NOT_IN_THE_TEXT;
  initnext(pat, m+1, next);                          /* Preprocess pattern */
  resume = next[m+1];
  next[m+1] = -1;
  j = k = 1;
  do {                                                          /* Search */
    if(j == 0 || text[k]==pat[j])
      {
        k++; j++;
      }
    else j = next[j];
    if(j > m)
      {
        Report_match_at_position(k-j+1); j := resume
      }
  } while(k ≤ n);
  pat[m+1] = END_OF_STRING;
}
```

Als obere Schranke für den Erwartungswert der Gesamtzahl der Vergleiche ergibt sich im Falle großer ($c > 10$) Alphabete:

$$\frac{\bar{C}_n}{n} \leq 1 + \frac{1}{c} - \frac{1}{c^m} \tag{8.4}$$

Im Vergleich zum naiven Algorithmus ergibt sich folgende Verringerung des Aufwands beim KMP-Algorithmus:

$$\frac{KMP}{naive} \approx 1 - \frac{2}{c^2} \tag{8.5}$$

8.4.1.5 Der Boyer-Moore-Algorithmus

Dieser effiziente Algorithmus vergleicht das Pattern von rechts nach links mit dem Text. Zusätzlich werden zwei Heuristiken zur Beschleunigung angewendet, nämlich eine Match-Heuristik (ähnlich wie beim KMP-Algorithmus) und eine Vorkommensheuristik.
Die **Match-Heuristik** verschiebt das Pattern so weit nach rechts, so daß an der neuen Vergleichsposition

1. Das Pattern alle vorher übereinstimmenden Zeichen matcht und
2. ein anderes Zeichen als vorher an der Vergleichsposition steht.

Beispiel zur Match-Heuristik:

```
..xaxraxxxxxxabracadabra

  a
  ra
    bra
            dabra
             abracadabra
```

Die Match-Heuristik wird als Tabelle `dd` implementiert. Hierbei gibt `dd[i]` den Betrag der Verschiebung der Vergleichsposition (nach rechts) im Text an, wenn beim $i$ten Zeichen des Patterns eine Ungleichheit auftrat. An der neuen Position beginnt der Vergleich mit dem letzten Zeichen des Patterns. Diese Tabelle ist wie folgt definiert:

$$\begin{aligned} dd[j] &= \min\{s+m-j | s \geq 1 \wedge ((s \geq j \vee pattern[j-s] \neq pattern[j]) \wedge \\ &\quad ((s \geq i \vee pattern[i-s] = pattern[i]) \quad \text{für } j < i \leq m))\} \end{aligned}$$

oder

$$\begin{aligned} dd[j] &= \min\{s+m-j | 1 \leq s \leq m \wedge \forall i\ 0 \leq i \leq m-j\ (s+i < j \rightarrow \\ &\quad (i = 0 \wedge pattern[j] \neq pattern[j-s] \vee i > 0 \wedge pattern[j] = pattern[j-s+i]))\} \end{aligned}$$

Beispiel: Tabelle `dd` *für das Pattern* `abracadabra`*:*

```
       a  b  r  a  c  a  d  a  b  r  a
dd[j] 17 16 15 14 13 12 11 13 12  4  1
```

Die **Vorkommensheuristik** verfolgt das Ziel, die Textposition, an der Ungleichheit auftrat, mit dem ersten übereinstimmenden Zeichen im Pattern zur Deckung zu bringen. Hierzu wird eine über das Textzeichen indizierte Tabelle `d` benutzt, wobei `d[c]`

analog zu `dd` den Betrag der Verschiebung der Vergleichsposition im Text angibt, wenn an der aktuellen Position das Zeichen `c` die Ungleichheit verursacht hat. Damit ergibt sich folgende Definition für `d`:

$$d[x] \quad = \quad \min\{s | s = m \vee (0 \leq s < m \wedge pattern[m-s] = x)\}$$

Beispiel: Tabelle `d` *für das Pattern* `abracadabra`*:*

$$d['a'] = 0 \quad d['b'] = 2 \quad d['c'] = 6 \quad d['d'] = 4 \quad d['r'] = 1 \tag{8.6}$$

(für alle anderen Zeichen `x` *ist* `d[x] = 11`*)*
Von den beiden Heuristiken wählt der Algorithmus jeweils den größeren Verschiebungsbetrag und zwar sowohl nach einer Ungleichheit wie auch nach einem Treffer. Die Implementierung in C sieht dann so aus:

```
bmsearch(text, n, pat,m)                    /* Search pat[1..m] in text[1..n] */
      char text[], pat[];
      int n, m;
{
  int k, j, skip;
  int dd[MAX_PATTERN_SIZE], d[MAX_ALPHABET_SIZE];

  initd(pat, m, d);                                  /* Preprocess the pattern */
  initdd(pat, m, dd);
  k = m; skip = dd[1] + 1;
  while(k ≤ n) {                                                  /* Search */
    j = m;
    while(j > 0 && text[k] == pat[j])
      {
        j--; k--;
      }
    if(j == 0)
      {
        Report_match_at_position(k+1);
        k += skip;
      }
    else
      k += max(d[text[k]], dd[j]);
  }
}
```

Bezüglich der Aufwandsabschätzung ergeben sich folgende Aussagen für den Boyer-Moore-Algorithmus:

- Im worst case ergibt sich $O(n + rm)$, wobei r die Anzahl Treffer angibt. Im ungünstigsten Fall ergibt sich damit der gleiche Aufwand wie beim naiven Algorithmus.
- Für große Alphabete und $m \ll n$ gilt die untere Schranke

$$\frac{\bar{C}_n}{n} \geq \frac{1}{m} + \frac{m(m+1)}{2m^2c} + O(c^{-2}) \tag{8.7}$$

- Bei ungleicher Auftretenswahrscheinlichkeit der Zeichen gilt $\bar{C}_n/n < 1$ unter der Voraussetzung

$$c\left(1 - \sum_{i=1}^{c} p_i^2\right) > 1. \tag{8.8}$$

8.4.1.6 Der Boyer-Moore-Horspool-Algorithmus

Dieser Algorithmus ist eine vereinfachte, beschleunigte Variante des Boyer-Moore-Algorithmus'. Er verwendet nur eine Vorkommensheuristik: Die Verschiebung der Vergleichsposition wird mit dem Zeichen des Textes berechnet, dessen Position momentan mit dem letzten Zeichen des Patterns korrespondiert. Das folgende Beispiel illustriert diese Vorgehensweise beim Pattern `abracadabra`:

```
..xaxxrbxdabracadabra

ra
    a
     a
       a
      adabra
              a
        abracadabra
```

Zusätzlich ist noch der Sonderfall zu behandeln, wenn das Textzeichen mit dem letzten Zeichen des Patterns übereinstimmt, aber weiter vorne eine Ungleichheit aufgetreten ist. Hierzu wird bei der Berechnung der Shift-Tabelle zuerst der Eintrag für das letzte Zeichen des Patterns auf den Wert m gesetzt und dann die Shift-Tabelle nur für die ersten $m - 1$ Zeichen des Patterns berechnet. Diese Tabelle ist wie folgt definiert:

$$d[x] \quad = \quad \min\{s | s = m \vee (1 \leq s < m \wedge pattern[m - s] = x)\}$$

Beispiel: Tabelle `d` *für das Pattern* `abracadabra`*:*

$$d['a'] = 3 \quad d['b'] = 2 \quad d['c'] = 6 \quad d['d'] = 4 \quad d['r'] = 1$$

(für alle anderen Zeichen `x` *ist* `d[x] = 11`*).*
Eine einfache Implementierung des Algorithmus' ist nachfolgend dargestellt:

```
bmhsearch(text, n, pat, m)                    /* Search pat[1..m] in text[1..n] */
      char text[], pat[];
      int n, m;
{
  int d[MAX_ALPHABET_SIZE], i, j, k;

  for(j=0; j<MAX_ALPHABET_SIZE; j++)
    d[j] = m;                                          /* Preprocessing */
  for(j=1; j<m; j++)
    d[pat[j]] = m-j;

  pat[0] = CHARACTER_NOT_IN_THE_TEXT;          /* To avoid having code */

  text[0] = CHARACTER_NOT_IN_THE_PATTERN;      /* for special cases */
  i = m;
  while(i ≤ n) {                                       /* Search */
    k = i;
    for(j=m; text[k] == pat[j]; j--) k--;
    if (j == 0)
      Report_match_atposition(k+1);
    i += d[text[i]];
  }  /* restore pat[0] and text[0] if necessary */
}
```

Der asymptotischer Aufwand dieses Algorithmus' für n und c (mit $c \ll n$ und $m > 4$) ist

$$\frac{\bar{C}_n}{n} = \frac{1}{m} + \frac{m+1}{2mc} + O(c^{-2}). \tag{8.9}$$

8.4.1.7 Der Shift-Or-Algorithmus

Dieser Algorithmus ist ein Echtzeit-Algorithmus, der ohne Zwischenspeicherung des Textes auskommt. Damit ist er besonders für eine hardwaremäßige Implementierung geeignet. Die Grundidee basiert hierbei auf der Theorie der endlichen Automaten: Es wird ein Vektor von m verschiedenen Zustandsvariablen benutzt, wobei die $i$te Variable den Zustand des Vergleichs zwischen den Positionen $1, \ldots, i$ des Patterns und den Positionen $(j-i+1), \ldots, j$ des Textes angibt (wobei j die aktuelle Textposition ist). Die Zustandsvariablen sind binär; die $i$te Zustandsvariable s_i hat dabei den Wert 0, falls die letzten i Zeichen übereinstimmen, und sonst den Wert 1. (Diese ungewöhnliche Definition von „Wahrheitswerten“ kann auch wieder umgedreht werden, dann muß aber u.a. im Algorithmus das OR durch ein AND ersetzt werden.)

Zweckmäßigerweise wird der Zustandsvektors *state* als Binärzahl repräsentiert:

$$state = \sum_{i=0}^{m-1} s_{i+1} \cdot 2^i \tag{8.10}$$

Ein Treffer endend an der aktuellen Position liegt dann vor, wenn $s_m = 0$ (bzw. $state < 2^{m-1}$).
Wenn ein neues Zeichen aus dem Text gelesen wird, muß der Statusvektor wie folgt modifiziert werden:

1. Der Vektor wird um 1 nach links verschoben und $s_1 = 0$ gesetzt.
2. Abhängig von dem eingelesenen Zeichen muß der Vektor aktualisiert werden. Hierzu dient eine Tabelle T mit Einträgen für jedes Zeichen des Alphabets. Der neue Statusvektor ergibt sich dann durch Oder-Verknüpfung des alten Vektors mit dem entsprechenden Tabelleneintrag.

Der modifizierte Statusvektor ergibt sich somit nach der Formel:

$$state = (state << 1) \text{ or } T[currchar] \tag{8.11}$$

(Hierbei steht << für einen Linksshift um eine Position.)
Beispiel für eine Tabelle T:
Alphabet: $\{a, b, c, d\}$ *Pattern: ababc*

```
T[a]=11010  T[b]=10101  T[c]=01111  T[d]=11111
```

Die exakte Definition der Tabelle T ist folgende:

$$T_x = \sum_{i=0}^{m-1} \delta(pat_{i+1} = x) \cdot 2^i \tag{8.12}$$

mit $\delta(C) = 0$, falls die Bedingung C erfüllt ist (sonst 1)
Beispiel für die Suche nach `ababc` *im Text* `abdababababc`*:*

```
Text :   a     b     d     a     b
T[x] : 11010 10101 11111 11010 10101
state: 11110 11101 11111 11110 11101

Text :   a     b     a     b     c
T[x] : 11010 10101 11010 10101 01111
state: 11010 10101 11010 10101 01111
```

Eine einfache Implementierung des Shift-Or-Algorithmus in C ist nachfolgend angegeben.

```
sosearch(text, n, pat, m)                    /* Search pat[1..m] in text[1..n] */
      register char *text;
      char pat[];
      int n, m;
{
  register char *end;
  register unsigned int state, lim;
  unsigned int T[MAXSYM], i, j;
  char *start;
  if(m > WORD)
    Abort(}Use pat size <= word size});

  for(i=0; i<MAXSYM; i++) T[i] = '0;                    /* Preprocessing */
  for(          /* Preprocessing */lim=0, j=1, i=1; i≤m; lim |= j, j ≪= B, i++)
      T[pat[i]] &= 'j;
  lim = '(lim ≫ B);
  text++; end = text+n+1;                                  /* Search */
  state = '0;                                            /* Initial state */
  for(start=text; text < end; text++)
    {
      state = (state ≪ B) | T[*text];                     /* Next state */
      if(state < lim)
        Report_match_at_position(text-start-m+2);
    }
}
```

Die Komplexität diese Algorithmus ist $O(\lceil \frac{m}{w} \rceil n)$, wobei $\lceil \frac{m}{w} \rceil$ den Aufwand zur Berechnung eines Shifts bzw. zur Oder-Verknüpfung von Bitstrings der Länge m angibt und w die Wortlänge des verwendeten Rechners ist.

8.4.1.8 Erweiterungen des Shift-Or-Algorithmus'

Im folgenden beschreiben wir die in [Baeza-Yates & Gonnet 92] vorgestellten Erweiterungen des Shift-Or-Algorithmus'.

8.4.1.8.1 Zeichenklassen

Wir verwenden hier folgende Symbole zur Spezifikation von Patterns:

- x bestimmtes Zeichen
- . beliebiges Zeichen
- $[Z]$ Zeichen aus der Menge Z
- $\overline{C}$ Komplementmenge der Klasse C

Zum Beispiel matcht das Pattern '$M[ae][ij].\overline{[g-ot-z]}$' die Zeichenfolgen 'Meier', 'Majer', 'Meise', aber nicht 'Maler' oder 'Maien'.
Zeichenklassen können im Shift-Or-Algorithmus einfach durch folgende geänderte Definition der Tabelle T behandelt werden:

$$T_x = \sum_{i=0}^{m-1} \delta(pat_{i+1} \in \text{Class}_{i+1}) \cdot 2^i$$

Somit ändert sich nur die Präprozessierung des Patterns, der eigentliche Algorithmus bleibt unverändert.

8.4.1.8.2 Zeichenkettensuche mit erlaubten Fehlern

Wir nehmen an, die Maximalzahl erlaubter Fehler sei vorgegeben. Im Algorithmus ersetzen wir dann die einzelnen Bits des Statusvektors durch einen Zähler für die Anzahl der Fehler und addieren die Einträge aus T anstelle der bisherigen OR-Verknüpfung. Das folgende Beispiel für die Suche nach `ababc` mit höchstens 2 Fehlern im String `abdababab c` illustriert diese Vorgehensweise:

text	:	a	b	d	a	b	a	b	a	b	c
$T[x]$	:	11010	10101	11111	11010	10101	11010	10101	11010	10101	01111
state	:	99990	99901	99121	92220	32301	34020	50301	14020	50301	04121
									*		*

Hier wird angenommen, daß die Zahl 9 für die größte darstellbare Zahl steht. Der eigentliche Algorithmus legt die Anzahl der Bits für die Zähler entsprechend der vorgegebenen Maximalzahl von Fehlern fest und verwendet einen zusätzlichen Bitvektor zur Markierung des Überlaufs.

8.4.1.8.3 Alternative Patterns

Hierbei wird gleichzeitig nach mehreren Patterns $p_1, \ldots, p_l$ gesucht. Dieser Fall läßt sich auf zwei Arten behandeln:

a) Mit einem eigenem Statusvektor und einer eigenen Tabelle T für jedes Pattern: Sei m_{max} die maximale Länge der Patterns, also $m_{max} = \max_i(|p_i|)$. Der Aufwand ist dann der l-fache gegenüber einem einzelnen Pattern, nämlich $O(\lceil \frac{m_{max}}{w} \rceil ln)$.

b) Alternativ dazu kann man alle Statusvektoren verketten und ebenso die entsprechenden Einträge in T. Sei $m_{sum} = \sum_i |p_i|$, dann ist der Aufwand nun $O(\lceil \frac{m_{sum}}{w} \rceil n)$, also i.d.R. niedriger.

8.4.1.9 Vergleich der Verfahren

Abbildung 8.11 zeigt experimentelle Ergebnisse für die Anwendung der vorgestellten Algorithmen auf englischsprachige Texte.

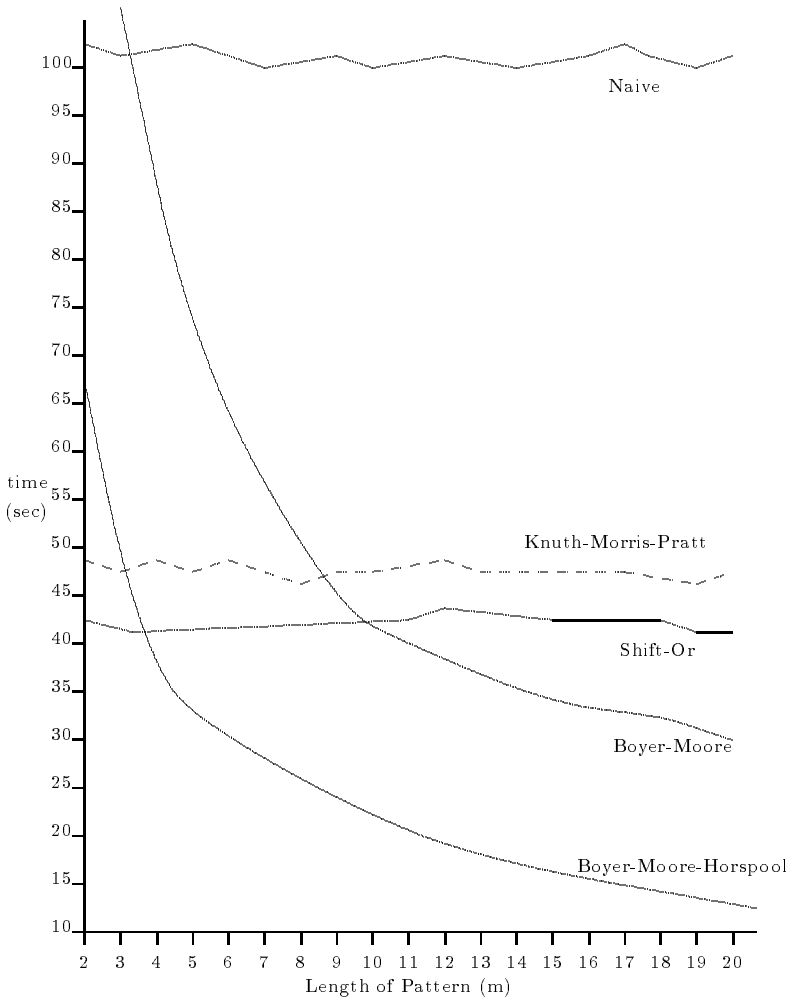


Abbildung 8.11: Experimentelle Ergebnisse für englischsprachigen Text

8.4.2 Ähnlichkeit von Zeichenketten

Die Ähnlichkeit von Zeichenketten spielt bei der die Suche nach Eigennamen und zur Korrektur von Tippfehlern eine wichtige Rolle. Wir stellen hier nur kurz drei wichtige Methoden vor; ein ausführlicherer Überblick findet sich in [Faloutsos 85].

8.4.2.1 Phonetisierungsalgorithmen

Für die Suche nach phonetisch gleichen Wörtern gibt es Phonetisierungsalgorithmen, die Wörter auf einen internen Code abbilden. Phonetisch gleichen Wörtern soll dabei möglichst der gleiche Code zugeordnet werden. Die Suche erfolgt dann auf der Basis dieser Codes. Zum Beispiel bildet der SOUNDEX-Algorithmus [Gadd 88] den gleichen Code für die englischsprachigen Wörter „Dixon", „Diksen" und „Dickson". Ein Nachteil dieses Verfahrens ist allerdings, daß ähnlich geschriebene Wörter häufig auf unterschiedliche Codes abgebildet werden, wie z.B. bei den Wörtern „Rodgers" und „Rogers".

8.4.2.2 Damerau-Levenstein-Metrik

Speziell zur Tippfehlerkorrektur wurde die Damerau-Levenstein-Metrik als Ähnlichkeitsmaß für Zeichenketten entwickelt. Diese Metrik soll beim Vergleich eines Textwortes mit einem Wort aus dem Wörterbuch die Zahl der Tippfehler annähern, um dann das ähnlichste Wort auszuwählen. Es werden vier mögliche Arten von Fehlern angenommen: Einfügung, Löschung, Substitution, Transposition. Für zwei Zeichenketten s und t wird dann die minimale Anzahl von Fehlern berechnet, mit der diese ineinander überführt werden können. Die DL-Metrik wird rekursiv berechnet, indem die beiden Strings zeichenweise verglichen werden, und bei Ungleichheit diejenige der vier Fehlerarten angenommen, die insgesamt den Abstand minimiert. Sei $f(i,j)$ der Abstand der zwei Zeichenketten, wenn bis zum $i$ten Zeichen von s und bis zum $j$ten Zeichen von t verglichen wurde. Der minimale Abstand berechnet sich dann nach der Formel:

$$f(0,0) = 0 \tag{8.13}$$

$$\begin{aligned} f(i,j) = \min\{&f(i-1,j)+1, f(i,j-1)+1, f(i-1,j-1)+d(s_i,t_j), \\ &f(i-2,j-2)+d(s_{i-1},t_j)+d(s_i,t_{j-1})+1\} \end{aligned} \tag{8.14}$$

Hierbei ist $d(s_i,t_j)$=0, falls das $i$te Zeichen von s und mit dem $j$ten Zeichen von t übereinstimmt (also $s_i = t_j$), sonst ist $d(s_i,t_j)$=1.Das nachfolgende Beispiel illustriert die Anwendung dieser Formel für den Vergleich der Zeichenketten MONSTER und CENTRE.

Zeichen		Operation	Kosten
M	C	Substitution	1
O	E	Substitution	1
N	N	=	0
S	–	Einfügung	1
T	T	=	0
E	R	halbe Transpos.	1/2
R	E	halbe Transpos.	1/2
		Summe	4

Wesentliche Nachteile der DL-Metrik sind zum einen die relativ aufwendige Berechnung, zum anderen das Fehlen geeigneter Zugriffspfade zur Beschleunigung der Suche in umfangreichen Wörterbüchern; lediglich durch Clustering kann hier eine gewisse Effizienzsteigerung erreicht werden.

8.4.2.3 Trigrams

Trigrams sind Zeichenfolgen der Länge 3. Bei der Ähnlichkeitssuche über Trigrams werden zunächst die Wörter auf die Menge der enthaltenen Trigrams abgebildet, also z.B. die Zeichenkette 'MENGE' auf {'_ME', 'MEN', 'ENG', 'NGE', 'GE_'}. Dann wird nach Wörtern gesucht, die in möglichst vielen Trigrams mit dem gegebenen Wort übereinstimmen. Somit hat man im Prinzip einen "coordination level match" auf der Ebene der Trigrams. Die Suche kann durch spezielle Zugriffspfade wie invertierte Listen oder Signaturen beschleunigt werden. Trigrams sind daher eine effiziente und wirkungsvolle Methode zur Ähnlichkeitssuche auf Zeichenketten.

8.4.3 Invertierte Listen

Die häufigste in IRS verwendete Zugriffsmethode sind invertierte Listen. Sie zeichnen sich durch extrem kurze Zugriffszeiten aus; nachteilig ist allerdings der Speicheroverhead und der hohe Änderungsaufwand, der eine sofortige Aktualisierung der invertierten Listen beim Einfügen oder Ändern von Dokumenten praktisch unmöglich macht.

8.4.3.1 Prinzipieller Aufbau

Eine invertierte Liste eines Terms ist eine aufsteigend sortierte Listen von Dokumentnummern derjenigen Dokumente, in denen der Term vorkommt, also z.B.:

t_1:	d_2	d_{15}	d_{23}	d_{89}	...
t_2:	d_5	d_{15}	d_{89}	...	

Mit invertierten Listen läßt sich boolesches Retrieval relativ einfach implementieren. Die einzelnen Operationen der booleschen Algebra werden dabei auf entsprechende Listenoperationen abgebildet:

$\vee$ Vereinigen der Listen
$\wedge$ Schneiden der Listen
$\wedge\neg$ Differenzbildung

Beispiel: Ergebnisliste für $t_1 \vee t_2$

d_2	d_5	d_{15}	d_{23}	d_{89}	...

Beispiel: Ergebnisliste für $t_1 \wedge t_2$

d_{15}	d_{89}	...

Soll auch die Suche mit Kontextoperatoren durch die invertierten Listen unterstützt werden, so müssen die Einträge für die einzelnen Dokumente entsprechend erweitert werden. Man nimmt dann Angaben über alle Vorkommen des Terms im Dokument in den Eintrag auf. Zum Beispiel kann für jedes einzelne Vorkommen die Kennung des entsprechenden Dokumentfeldes, die Satznummer und die Wortnummer festgehalten werden; dadurch ist es möglich, Bedingungen bezüglich des Dokumentfeldes (z.B. Vorkommen nur im Titel), Wortabstand, Wortreihenfolge und des Vorkommens im gleichen Satz auf den invertierten Listen abzuprüfen, ohne auf das eigentliche Dokument zugreifen zu müssen. Diese enorme Beschleunigung der Suche muß allerdings mit einem hohen Speicherplatzbedarf erkauft werden: Invertierte Listen, die nur boolesche Operatoren unterstützen, begnügen sich bei geschickter Organisation mit 10 % des Speicherplatzes der Primärdaten, wohingegen die Listen mit erweiterten Einträgen bis zu 100 % benötigen.
Der effektiv belegte Speicherplatz hängt auch von der Verteilung der invertierten Listen auf die einzelnen Blöcke des Sekundärspeichers ab. In den meisten Darstellungen zu diesem Thema (z.B. [Faloutsos 85]) geht man von einer kompakten Speicherung aus, wo also die Menge der zur Verfügung stehenden Blöcke als ein großer linearer Speicher betrachtet wird, in dem die einzelnen Listen direkt aufeinanderfolgend gespeichert sind. Eine solche Organisation macht aber die Änderung der Listen praktisch unmöglich; daher ist sie nur für Datenbanken auf CD-ROM geeignet. Häufig geht man davon aus, daß nur Dokumente am Ende der Datenbasis hinzugefügt werden, während das Löschungen oder Änderungen fast nie vorkommen (evtl. kann man einfach die entsprechenden Einträge als ungültig markieren). In diesem Fall werden nur am Ende der Liste neue Einträge hinzugefügt; dies läßt sich relativ einfach dadurch realisieren, daß man die Speicherblöcke zu Beginn nicht zu 100 % füllt (da es viele seltene Terms, aber wenige häufige gibt, muß man bei kurzen invertierten Listen mehrere davon in einen einzigen Block packen). Außerdem sollte jede Liste auf möglichst wenige Blöcke verteilt werden, also Listen, die kürzer sind als ein Block, stets in einem einzigen Block ablegen, und längere Listen sollten stets an einer Blockgrenze beginnen.
Eine zu invertierten Listen ähnliche Speicherungsstruktur sind B*-Bäume. Diese eignen sich allerdings für Textretrieval weniger: Sinnvollerweise müßte man für alle Terms einen einzigen B*-Baum anlegen; ein einziger Eintrag müßte dann sowohl den Term als auch die Dokumentnummer enthalten, was ungefähr zu einer Verdoppelung des

Speicherplatzbedarfs führen würde. Der einzige Vorteil dieser Lösung wäre die leichtere Änderbarkeit bei häufigen Änderungen im Dokumentenbestand. In [Motzkin 91] wird ein zweistufiger B*-Baum beschrieben, der die Vorteile des B*-Baums mit denen invertierter Listen verbindet.

8.4.3.2 Ranking mit invertierten Listen

Oben wurde bereits gezeigt, wie einfach man boolesches Retrieval mit invertierten Listen implementieren kann. Ein häufiges Vorurteil gegenüber Rankingverfahren ist der wesentlich höhere Berechnungsaufwand zur Bestimmung der Rangliste der Dokumente. In diesem Abschnitt beschreiben wir nun Algorithmen zur Bestimmung der k Dokumente mit dem höchsten Retrievalgewicht, wobei k von außen vorgegeben ist. Dabei gehen wir von folgenden Annahmen aus:

- Als Retrievalfunktion soll das Skalarprodukt berechnet werden. (Andere Retrievalfunktionen lassen sich durch geringfügige Modifikation der beschriebenen Algorithmen realisieren.)
- Die Einträge in der invertierten Liste enthalten zusätzlich das Indexierungsgewicht des Terms bzgl. des betreffenden Dokumentes.

Das Ziel beim Entwurf effizienter Algorithmen für diese Aufgabenstellung muß die Minimierung der Anzahl der Plattenzugriffe sein. Daher muß die Berechnung der Retrievalgewichte ausschließlich über die invertierten Listen erfolgen. Der in [Salton 88, S. 248ff.] beschriebene Algorithmus, der für jedes in den invertierten Listen neu auftretende Dokument den Zugriff auf den zugehörigen Dokumentvektor (und damit einen zusätzlichen Plattenzugriff) erfordert, würde daher zu unzumutbaren Wartezeiten führen: Bei einer mittleren Zugriffszeit von 15 ms bei heutigen Plattenlaufwerken kann man pro Sekunde nur 66 Dokumentnummern abarbeiten! Demgegenüber hat man schon bei Blöcken von 2KB und 6 Bytes pro Eintrag (3–4 Bytes für die Dokumentnummer und 2 Bytes für das Indexierungsgewicht) mehr als 300 Dokumentnummern pro Block; außerdem werden die Blöcke einer invertierten Liste sequentiell gelesen, was schneller als ein wahlfreier Zugriff ist.
Im folgenden werden zwei Algorithmen vorgestellt, die die oben gestellten Anforderungen erfüllen. Dabei ist der zuerst beschriebene naive Algorithmus in erster Linie zur Illustration der prinzipiellen Vorgehensweise gedacht; bei einer praktischen Implementierung würde man eher eine Variante des zweiten Algorithmus (von Buckley und Lewit) bevorzugen.

8.4.3.2.1 Naiver Algorithmus

Dieser Algorithmus führt im Prinzip ein Mischen aller invertierten Listen der Frageterms aus, was einer ODER-Verknüpfung dieser Terms beim booleschen Retrieval

entspricht. Zusätzlich werden beim Mischen die Retrievalgewichte berechnet. Da wir nur an den obersten k Dokumenten interessiert sind, wird aber nicht die vereinigte Liste explizit erzeugt, sondern es werden nur die k Dokumente mit dem höchsten Gewicht festgehalten.

In dem nachfolgenden Programm werden folgende Parameter verwendet:

`t`	Anzahl der Frageterme,
`terms[]`	Vektor mit den Nummern der Frageterme,
`qw[]`	Vektor mit den Fragetermgewichten,
`k`	gewünschte Anzahl Dokumente,
`top_doc[]`	Vektor mit den gefundenen Dokumentnummern,

Zusätzlich nehmen wir an, daß eine Funktion `readil(term,&(wt))` existiert, die jeweils den nächsten Eintrag aus der invertierten Liste des Terms `term` liest und als Funktionswert die Dokumentnummer und in `wt` das Indexierungsgewicht liefert. Beim ersten Aufruf für einen Term wird die invertierte Liste geöffnet, und am Ende der Liste wird die Dokumentnummer `MAXINT` und das Indexierungsgewicht 0 zurückgeliefert.

```
naivematch(t,terms,qw,k,top_doc)
      int t, k;
      int terms[], top_doc[];
      float qw[];
{
  int i, j, l, mind;
  int ild[MAX_QT], it[MAX_QT];
  float s;
  float ilw[MAX_QT], sim[MAX_TOPDOC];

  for (l=1; l≤t; l++) it[l] = l;
  do
    {
      for (i=1; i<l; i++)
      ild[it[i]] = readil(terms[it[i]],&(ilw[it[i]]));
      mind = MAXINT;
      for (i=1; i≤t; i++)
        {
          if (ild[i]<mind) {
            s = ilw[i] * qw[i];
            mind = ild[i];
            it[1] = i;
            l = 2;
          }
          else if (ild[i]==mind) {
```

```
            s += ilw[i] * qw[i];
            it[l++] = i;
          }
          if (s>sim[k])
            {
              ...sim und top_doc aktualisieren
              }
        }
    }
  while (mind ≤ MAXINT);
}
```

8.4.3.2.2 Algorithmus von Buckley & Lewit

Dieser in [Buckley & Lewit 85] beschriebene Algorithmus berechnet die Retrievalgewichte iterativ durch Term-weises Vorgehen. Hierzu wird ein interner Vektor zum Speichern der Retrievalgewichte aller Dokumente der Kollektion verwendet. (In [Lucarella 88] wird eine Variante dieses Algorithmus' vorgeschlagen, bei der durch eine dynamische Verwaltung die Größe dieses Vektors deutlich reduziert werden kann.) Die prinzipielle Vorgehensweise ist nun wie folgt:

- Betrachte die invertierten Listen in der Reihenfolge fallender Fragetermgewichte
- Für jedes Paar (Dokumentnummer, Indexierungsgewicht) in der invertierten Liste eines Terms berechne dessen Beitrag zum Retrievalgewicht des Dokumentes und addiere ihn zum bisherigen Gewicht. Halte die bisher höchsten $k+1$ Retrievalgewichte (und die zugehörigen Dokumentnummern) fest.
- Nach dem Ende jeder invertierten Liste wird geprüft, ob es noch möglich ist, daß das $k+1$te Dokument den Rang k erreichen kann. Ist dies nicht der Fall, wird der Algorithmus beendet.

```
optmatch(t,terms,qw,k,top_doc)
      int t, k;
      int terms[], top_doc[];
      float qw[];
{
  int i, ild;
  float ilw;
  float sim[NUM_DOCS];
  /* ... qw und terms nach fallenden Gewichten ordnen
     ... sim und top_doc initialisieren */
```

```
  for (i=1; i≤t; i++)
    {
      ild = readil(terms[i],&ilw);
      while (ild<MAXINT)
        {
          sim[ild] += qw[i]*ilw;
          if (sim[ild] > sim[top_doc[k+1]])
            {                                    /* ... top_doc aktualisieren */
            }
          ild = readil(terms[i],&ilw);
        }
      /* ... Setze max_remaining_sim auf das maximale
         Retrievalgewicht, das top_doc[k+1] fuer die
         restlichen Terms erreichen kann */
      if (sim[top_doc[k+1]]+max_remaining_sim ≤ sim[top_doc[k]])
        return;
    }
}
```

Die Berechnung von `max_remaining_sim` hängt von der Art der Indexierungsgewichte ab. In [Buckley & Lewit 85] werden hierzu verschiedene Fälle diskutiert (z.B. normierte Dokumentvektoren). Wir betrachten hier nur den Fall, daß für die Indexierungsgewichte eine feste obere Schranke S existiert (z.B. $S = 1$ bei probabilistischen Indexierungsgewichten). Damit erhält man:

$$\texttt{max\_remaining\_sim} = S * \sum_{j=i+1}^{t} \texttt{qw[j]} \tag{8.15}$$

Der Algorithmus läßt sich noch weiter beschleunigen, wenn man anstelle der korrekten Bestimmung der k Dokumente mit den höchsten Retrievalgewichten nur die obersten n Dokumente (mit $n < k$) exakt bestimmt, aber die ersten k Dokumente zurückliefert. In diesem Fall muß man die beiden letzten Zeilen in obigem Programm nur geringfügig ändern:

```
if (sim[top_doc[k+1]]+max_remaining_sim
        <= sim[top_doc[n]]) return;
```

Die Tabelle 8.2 zeigt experimentelle Ergebnisse von Buckley & Lewit für die Anwendung des vorgestellten Algorithmus' auf die INSPEC-Kollektion (84 Fragen, 12684 Dokumente). Als Qualitätsmaß wurde hier der Recall nach 10 Dokumenten betrachtet. Man erkennt, daß die Wahl eines kleineren n nur geringfügigen Einfluß auf die Retrievalqualität hat, aber der Berechnungsaufwand deutlich reduziert wird.

n	CPU-Zeit	#gel. Blöcke	Recall
1	61.3	2833	.1588
2	68.2	3058	.1566
3	75.9	3255	.1555
4	83.1	3451	.1538
5	90.3	3630	.1548
6	93.6	3721	.1543
7	99.7	3863	.1501
8	104.5	3980	.1505
9	107.0	4033	.1508
10	109.2	4111	.1508

Tabelle 8.2: Experimentelle Ergebnisse von Buckley & Lewit für die INSPEC-Kollektion

8.4.4 Signaturen

8.4.4.1 Das Signaturkonzept

8.4.4.1.1 Grundidee

Signaturen sind Bitstrings fester Länge, auf die man im IR Wörter und Texte abbildet. (Man kann Signaturen auch in Standard-Datenbanksystemen für andere Arten von Attributwerten einsetzen.) Indem man nun die Suchoperationen auf den Signaturen durchführt, erhält man eine deutliche Effizienzsteigerung gegenüber der Suche in den ursprünglichen Texten. Durch spezielle Speicherungsformen für die Signaturen ist eine weitere Beschleunigung möglich.
Im folgenden verstehen wir unter einer Signatur einen Bitstring S mit

$$S := < b_1, b_2, \ldots, b_L > \quad \text{mit} \quad b_i \in \{0,1\}, L \in I\!N \tag{8.16}$$

Signaturen werden durch die Abbildung von Wörtern auf Bitstrings erzeugt; diese Abbildung wird i.a. mit Hilfe von Hash-Funktionen definiert. Dabei entstehen sogenannte **Homonyme**, wenn verschiedene Wörter auf die gleiche Signaturen abgebildet werden. Da somit keine eineindeutige Abbildung zwischen Wörtern und Signaturen existieren, können Signaturen nur als Filter eingesetzt werden, um eine Obermenge der gesuchten Wörter bzw. Dokumente zu bestimmen (siehe 8.4.4.1.2).
Man unterscheidet zwei Arten von Signaturen:

- Bei **Binärsignaturen** werden die Wörter auf alle 2^L möglichen Signaturen abgebildet. Der zugehörige Signaturoperator $=_S$ prüft dann die Gleichheit von Anfrage- und Satzsignatur, um mögliche Treffer zu bestimmen.
- Bei **überlagerungsfähigen Signaturen** wird der Wert einer Signatur nur durch die gesetzten Bits bestimmt. Als **Signaturgewicht** g bezeichnet man

dabei die Anzahl der gesetzten Bits, die i.a. für alle Wörter gleich ist. Somit werden die Wörter hier auf $\binom{L}{g}$ verschiedene Signaturen abgebildet. Der wesentliche Vorteil überlagerungsfähiger Signaturen besteht nun darin, daß man neben einzelnen Wörtern auch ganze Texte auf eine einzige Signatur abbilden kann, indem man die Signaturen der im Text vorkommenden Wörter mit ODER verknüpft, wie im nachfolgenden Beispiel:

text	010010001000	S_1
search	010000100100	S_2
methods	100100000001	S_3
text search methods	110110101101	$S_1 \vee S_2 \vee S_3$

Im Vergleich zu Binärsignaturen ergeben sich folgende Vor- und Nachteile überlagerungsfähiger Signaturen:

- – Durch die Überlagerung entstehen zusätzliche Fehler (sogenannte **Phantome**), da die gesetzten Bits nicht mehr eindeutig den Ausgangssignaturen zugeordnet werden können.
- \+ Es ist wesentlich einfacher, geeignete Indexstrukturen für die Signaturen anzulegen.
- \+ Man kann ganze Textblöcke (Mengen von Wörtern) auf eine einzige Signatur abbilden, was zu einfacheren und effizienteren Suchalgorithmen führt.

Zum Vergleich von Anfrage- und Satzsignatur bei überlagerungsfähigen Signaturen dient der Signaturoperator $\supseteq_S$. Dieser prüft das Enthaltensein der Anfragesignatur in einer Satzsignatur, also

$$S \supseteq_S S^Q \Leftrightarrow (\forall)((1 \leq i \leq L) \wedge ((b_i^Q = 1) \Rightarrow (b_i = 1))),\ S, S^Q \in S_L. \qquad (8.17)$$

Dieser Operator kann auf effiziente Bitoperationen zurückgeführt werden, wie sie von allen Rechnerarchitekturen angeboten werden:

$$S \supseteq_S S^Q \Leftrightarrow S \wedge S^Q = S^Q \Leftrightarrow (\neg S) \wedge S^Q = 0_S \qquad (8.18)$$

Das folgende Beispiel illustriert die Anwendung dieses Operators:

text search methods	110110101101
in search of knowledge-based IR	010110101110
an optical system for full text search	010110101100
the lexicon and IR	101001001001
Anfrage:	
text search	010010101100

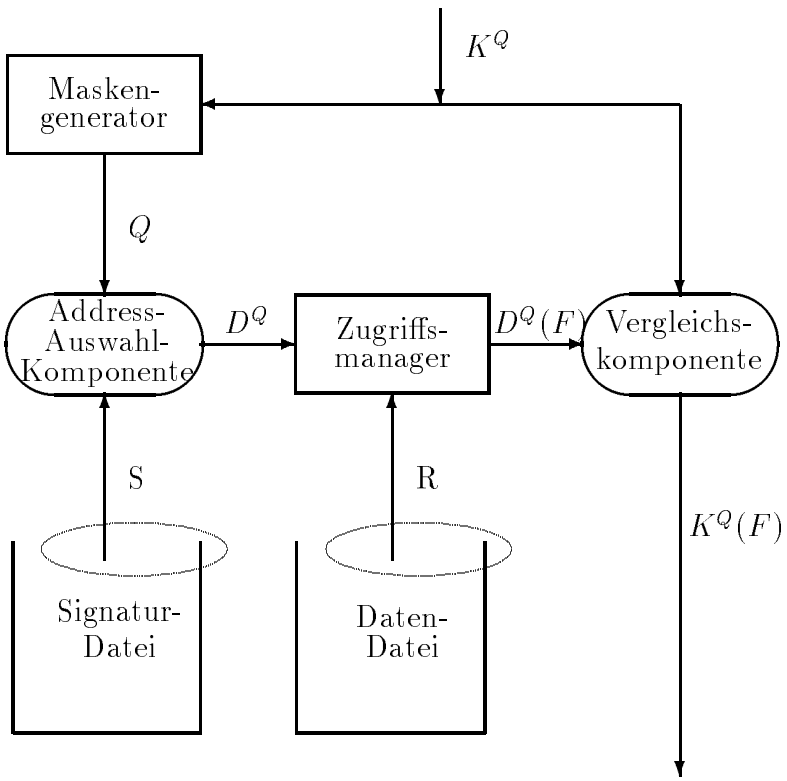


Abbildung 8.12: Organisation eines Signatur-Systems

Der Signaturoperator liefert die ersten drei Datensätze als Antworten. Ein Blick auf die zugehörigen Texte zeigt, daß der zweite Satz die gesuchten Wörter nicht enthält. Solche Antworten bezeichnet man als **false drops**. Sie entstehen durch Homonyme und Phantome. Ein wesentliches Ziel beim Entwurf von Signaturen ist daher, die Anzahl der false drops nicht über eine vorgegebene Schwelle steigen zu lassen (siehe folgende Abschnitte).

Im folgenden betrachten wir nur noch überlagerungsfähige Signaturen.

8.4.4.1.2 Prinzipielle Organisation eines Signatur-Systems

Abb. 8.12 zeigt die prinzipielle Organisation eines Signatur-Systems. Für eine Anfrage K^Q wird zunächst im Maskengenerator die zugehörige Signatur Q erzeugt. Die Signaturdatei enthält die Signaturen aller Datensätze. In der Adreßauswahlkomponente werden diese Signaturen mit der Anfragesignatur verglichen. Als Ergebnis erhält man eine Menge D^Q von Adressen von Datensätzen, deren Signaturen den Test bestanden haben. Der Zugriffsmanager holt nun die zugehörigen Datensätze $D^Q(F)$ aus der

Datendatei. In der Vergleichskomponente werden diese Datensätze mit der ursprünglichen Anfrage verglichen, um die false drops auszufiltern.

8.4.4.2 Codierungsmethoden

Wir betrachten nun verschiedene Methoden, um Mengen von Wörtern bzw. n-grams auf Signaturen abzubilden

8.4.4.2.1 Disjoint Coding

Beim disjoint coding (das man auch word coding nennt, wenn es wie hier auf Wörter angewendet wird), wird jedes Wort einzeln auf eine Signatur abgebildet. Die Signatur eines Textes besteht dann aus der Verkettung der Signaturen der einzelnen Wörter. Wir führen nun folgende Symbole ein:

L Länge der Signatur,
g Signaturgewicht (Anzahl gesetzter 1-Bits),
$SP = SP(L,g)$: Signaturpotential (Anzahl verschiedener erzeugbarer Kodierungen)

Man versucht nun Signaturen so zu entwerfen, daß das Signaturpotential für eine vorgegebene Länge L maximiert wird; dadurch erreicht man eine minimale Fehlerrate (s.u.). Das Signaturpotential berechnet sich aus der Länge und dem Gewicht zu:

$$SP = \left(\begin{array}{c} L \\ g \end{array} \right) = \frac{L!}{g!(L-g)!} \tag{8.19}$$

Dieser Wert ist maximal für $g = \frac{L}{2}$. Zum einen weiß man aus der Informationstheorie, daß ein Datenstrom dann die maximale Information enthält, wenn die Wahrscheinlichkeit für ein gesetztes Bit $\frac{1}{2}$ ist. Hier soll aber auch kurz der Beweis für unsere konkrete Anwendung skizziert werden:
Da $\binom{L}{g} = \binom{L}{L-g}$, können wir unsere Betrachtungen auf den Fall $g \leq \lfloor \frac{L}{2} \rfloor$ beschränken. Nun sei

$$SP_1 = SP(L, \lfloor \tfrac{L}{2} \rfloor) \quad \text{und}$$

$$SP_2 = SP(L, \lfloor \tfrac{L}{2} \rfloor - 1) = SP_1 \cdot \frac{g}{L-(g-1)}.$$

Wegen $g \leq \frac{L}{2}$ folgt daraus:

$$SP_2 \leq SP_1 \cdot \left(\frac{\left\lfloor \frac{L}{2} \right\rfloor}{\left\lfloor \frac{L}{2} \right\rfloor + 1} \right) < SP_1$$

Die gewünschte Aussage beweist man dann durch Induktionsbeweise über g und über L.
Wir betrachten nun die **Fehlerrate** beim disjoint coding. Hierzu führen wir folgende Bezeichnungen ein:

F Fehlerrate = Anzahl zu erwartender Fehler (fälschlicherweise gefundene Signaturen)
W Wörterbuchgröße (Anzahl verschiedener Wörter)
N Anzahl Sätze in der Datenbank

Wenn den Wörtern Signaturen zugeordnet werden, dann werden W verschiedene Wörter auf $SP = \binom{L}{\lfloor \frac{L}{2} \rfloor}$ verschiedene Signaturen abgebildet. Somit sind einer Signatur im Mittel $\frac{W}{SP}$ Wörter zugeordnet. Wird nach einem bestimmten Wort in der Datenbank gesucht, dann erhält man auch die $\frac{W}{SP} - 1$ Signaturen zu den anderen Wörtern als Antworten. Die erwartete Fehlerrate ist somit

$$F = \left(\frac{W}{SP} - 1\right) \frac{N}{W}. \tag{8.20}$$

Bei der Anwendung von Signaturen möchte man für einen bestimmten Datenbestand die Signaturlänge so wählen, daß eine vorgegebene Fehlerrate nicht überschritten wird. Aus vorstehender Formel 8.20 erhält man für das Signaturpotential als Funktion der Fehlerrate, der Wörterbuchgröße und des Datenvolumens:

$$SP = \frac{W \cdot N}{F \cdot W + N}. \tag{8.21}$$

Über $SP = \binom{L}{g}$ kann man daraus die Signaturlänge bestimmen. Tabelle 8.3 zeigt einige Signaturpotentiale hierzu.

L	g	SP
8	4	70
16	8	12 870
24	12	2 704 156
32	16	601 080 390

Tabelle 8.3: Einige Signaturpotentiale beim disjoint coding

Um eine bessere Vergleichbarkeit mit dem im nächsten Abschnitt beschriebenen block superimposed coding zu gewährleisten, betrachten wir hier noch den Fall der **blockweisen Codierung von Wörtern**. Statt für einen Text die Wortsignaturen entsprechend der Wortreihenfolge zu verketten (dies wäre sinnvoll, wenn man Kontextoperatoren unterstützen möchte), unterteilen wir den Text zunächst in Blöcke und bilden dann die *Menge* der im Block enthaltenen Wörter auf die Signatur ab. Sei B die Gesamtzahl der Blöcke in der Datenbank und w die Anzahl der (verschiedenen) Wörter pro Block. Wir möchten nun die zugehörige **Fehlerrate bei blockweiser Codierung** wissen. Hierzu nehmen wir an, daß sich die x Vorkommen (Token) eines Wortes zufällig über die B Blöcke verteilen. Der Erwartungswert $E(V(x))$ für die

Anzahl Blöcke, in denen das Wort auftritt, ist dann

$$E(V(x)) = B\left(1 - \left(1 - \frac{1}{B}\right)^{x}\right). \tag{8.22}$$

($1 - 1/B$ ist die Wahrscheinlichkeit, daß ein Token nicht in einem bestimmten Block auftritt, und diese Wahrscheinlichkeit mit x potenziert ist die W., daß keines der x Token im Block auftritt; die Gegenwahrscheinlichkeit, daß mindestens ein Token im Block auftritt, multipliziert mit der Anzahl aller Blöcke, ist dann der gesuchte Erwartungswert.) Auf eine Signatur entfallen nun im Mittel $B\frac{w}{SP}$ Token. Um nun für eine erfolglose Anfrage die Fehlerrate zu bestimmen, setzen wir in Gleichung 8.22 $x = B\frac{w}{SP}$ ein, um als Fehlerrate die Anzahl Blöcke zu erhalten, die sich auf die Anfrage qualifizieren:

$$\begin{aligned} F &\approx B\left(1 - \left(1 - \frac{1}{B}\right)^{B\frac{w}{SP}}\right) \\ &\approx B\left(1 - \exp\left(-\frac{w}{SP}\right)\right) \\ &\approx B\frac{w}{SP} \end{aligned}$$

Daraus erhalten wir die **Fehlerwahrscheinlichkeit**:

$$\begin{aligned} f &\approx 1 - \exp\left(-\tfrac{w}{SP}\right) \\ &\approx \frac{w}{SP} \end{aligned}$$

8.4.4.2.2 Block Superimposed Coding

Beim block superimposed coding werden die Signaturen aller Wörter, die in einem Textblock auftreten, zu einer einzigen Satzsignatur überlagert. Da nun ganze Texte durch eine einzige Signatur repräsentiert werden können, vereinfacht sich die Suche; anstelle einer Iteration über alle Wortsignaturen eines Blocks beim disjoint coding ist nunmehr ein einziger Vergleich mit der Satzsignatur notwendig.
Folgende Notationen werden nachstehend verwendet:

- L Länge der Signatur,
- g Gewicht (= Anzahl gesetzter Bits) für ein einzelnes Wort,
- λ Anzahl überlagerte Wortsignaturen,
- t Anzahl gesetzter Bits in der überlagerten Signatur,
- $\bar{a}$ mittlere Anzahl Wörter in einer Anfrage,
- γ durchschnittliches Signaturgewicht einer Anfrage,

F Anzahl false drops,
N Anzahl Satzsignaturen,
$f(t) = F/N$:Fehlerwahrscheinlichkeit.

Zur Abschätzung der **Fehlerwahrscheinlichkeit** wollen wir nun die Wahrscheinlichkeit bestimmen, daß eine zufällige Signatur sich für eine Anfragesignatur qualifiziert. Da eine Satzsignatur (ebenso wie eine Anfragesignatur) hier durch die Überlagerung von mehreren Wortsignaturen entsteht, berechnen wir zunächst die Wahrscheinlichkeit, daß durch Überlagerung von λ Wortsignaturen der Länge L mit Gewicht g eine Signatur entsteht, die an t bestimmten Stellen eine 1 enthält. Diese ist gleich der Wahrscheinlichkeit, daß zu einer Anfragesignatur mit t gesetzten Bits eine Satzsignatur existiert, die ebenfalls diese t 1-Bitpositionen besitzt. Die exakte Formel hierzu lautet [Roberts 79]:

$$p(L,g,\lambda,t) = \sum_{j=1}^{t} (-1)^j \binom{t}{j} \left(\frac{\binom{L-j}{g}}{\binom{L}{g}} \right)^{\lambda} \tag{8.23}$$

Für kleine t und λ läßt sich diese Formel annähern durch:

$$p(L,g,\lambda,t) \approx [p(L,g,\lambda,1)]^t = (1-(1-\frac{g}{L})^{\lambda})^t \tag{8.24}$$

Die Herleitung dieser Approximation kann man daduch veranschaulichen, daß $\frac{g}{L}$ für die Wahrscheinlichkeit einer bestimmten 1-Bitposition einer Anfragesignatur steht, $1-\frac{g}{L}$ daher für die einer 0. Durch Potenzieren mit λ erhält man die Wahrscheinlichkeit für das Verbleiben einer 0 nach dem Verodern von λ Attributsignaturen, durch Subtraktion dieses Ergebnisses von 1, daß eine bestimmte Bitposition der Satzsignatur gesetzt ist. Aus dem folgenden Potenzieren mit t resultiert die näherungsweise Wahrscheinlichkeit für t bestimmte 1-Bitpositionen.

Da man nicht im voraus bestimmen kann, welche der gefundenen Kandidaten zu false drops und welche zu korrekten Antworten führen werden, gehen wir vom pessimistischen Standpunkt aus und nehmen an, daß sich kein Treffer unter den Datensätzen befindet. Unter dieser Voraussetzung gibt $p(t)$ auch gleichzeitig die Fehlerwahrscheinlichkeit $f(t)$ an. Da, mit F als Fehlerrate und N als Anzahl vorhandener Datensätze, $f(t) = F(t)/N$ gilt, kann man durch Umstellung der Gleichung nach $F(t)$ die voraussichtliche Anzahl der false drops berechnen.

Im Normalfall wird man an das Problem der Signaturkodierung mit einer Vorstellung über die noch tolerierbare Fehlerrate in Abhängigkeit von der Anzahl spezifizierter Attribute herangehen und sucht von diesem Punkt ausgehend die optimale Signaturlänge L mit dem zugehörigen Gewicht g der Attributsignaturen. Dazu setzt man das durchschnittliche Signaturgewicht einer Anfrage mit $\bar{a}$ spezifizierten Attributen

$$\gamma = \sum_{i=1}^{L} p(1) = L \cdot \left(1 - \left(1 - \frac{g}{L}\right)^{\bar{a}}\right) \tag{8.25}$$

in die Approximation der Fehlerwahrscheinlichkeit

$$f \approx \left(1 - \left(1 - \frac{g}{L}\right)^{\lambda}\right)^{\gamma} \tag{8.26}$$

ein. Nach Umformung zu

$$\ln f \approx L \cdot \left(1 - \left(1 - \frac{g}{L}\right)^{\bar{a}}\right) \cdot \ln\left(1 - \left(1 - \frac{g}{L}\right)^{\lambda}\right)$$

und Ersetzung von

$$\left(1 - \frac{g}{L}\right)^{\bar{a}} \quad \text{durch} \quad \exp\left(\frac{\bar{a}}{\lambda} \cdot \ln\left(1 - \frac{g}{L}\right)^{\lambda}\right) \tag{8.27}$$

kann man $y = \left(1 - \frac{g}{L}\right)^{\lambda}$ setzen und erhält

$$\ln f \approx L \cdot \left(1 - \exp\left(\frac{\bar{a}}{\lambda} \ln y\right)\right) \cdot \ln(1 - y) \tag{8.28}$$

Um die nun folgende Minimierung dieser Funktion zu vereinfachen, setzen wir

$$\mathrm{e}^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \quad \text{mit} \quad x = \frac{\bar{a}}{\lambda} \tag{8.29}$$

Von dieser Potenzreihe benötigen wir für eine realistische Abschätzung lediglich die ersten beiden Glieder, da $\frac{\bar{a}}{\lambda} \ll 1$ ist und die nachfolgenden Glieder relativ schnell vernachlässigbar klein werden.[3] Daher bleibt

$$\ln f \approx -L \cdot \frac{\bar{a}}{\lambda} \cdot \ln y \cdot \ln(1 - y)$$

als zu optimierende Funktion. Betrachtet man L, $\bar{a}$ und λ als Konstanten, dann findet man das gesuchte Minimum bei $y = \frac{1}{2}$. Einsetzen von $f = \frac{F}{N}$ und $y = \frac{1}{2}$ liefert

$$\ln\left(\frac{F}{N}\right) \approx -L \cdot \frac{\bar{a}}{\lambda} \cdot \left(\ln \frac{1}{2}\right)^2$$

[3] Das entspricht bei Verwendung der durch die Gleichungen 8.30 und 8.31 berechneten optimalen Werte für L und g der Näherung $2^{-\frac{\bar{a}}{\lambda}} \approx 1 + \ln 2^{-\frac{\bar{a}}{\lambda}}$, die schon bei großen $\frac{\bar{a}}{\lambda}$ annehmbar ist.

und somit die optimale Signaturlänge

$$L_{opt} = \frac{\lambda}{\bar{a} \cdot \ln 2} \cdot \log_2 \frac{N}{F} \tag{8.30}$$

Aus $y = \frac{1}{2} = \left(1 - \frac{g}{L}\right)^{\lambda}$ resultiert schließlich das **optimale Wortgewicht**

$$g_{opt} = L \cdot \left(1 - 2^{-\frac{1}{\lambda}}\right). \tag{8.31}$$

Daß man mit g_{opt} richtig liegt, sieht man, indem man den Erwartungswert des Satzsignaturgewichts Γ aus den Wortsignaturen ermittelt:

$$\Gamma = \sum_{i=1}^{L} p(g_{opt}) = L \cdot \left(1 - \left(1 - \frac{g_{opt}}{L}\right)^{\lambda}\right) = \frac{L}{2}$$

Denn $\frac{L}{2}$ ist das optimale Satzsignaturgewicht nach Überlagerung von λ Wortsignaturen mit optimalem Gewicht g_{opt}.
Wir betrachten nun die Abhängigkeit der Fehlerwahrscheinlichkeit von der Anzahl Wörter in der Anfrage. Setzt man g_{opt} in die Ausgangsformel 8.26 für die Fehlerwahrscheinlichkeit ein, so bekommt man die Gleichungen

$$f(t) = 2^{-t} \quad \text{und} \quad F(t) = N \cdot 2^{-t}$$

wobei für a_q Wörter in der Anfrage $t = g(a_q)$ Bits gesetzt sind:

$$t = g(a_q) = \frac{\lambda}{\bar{a} \cdot \ln 2} \cdot \log_2 \frac{N}{F} \cdot \left(1 - 2^{-\frac{a_q}{\lambda}}\right).$$

Für $a_q = \bar{a}$ erhalten wir die gewünschte Fehlerrate $f(g(a_q)) = F/N$. Wesentlich wichtiger ist aber der allgemeineZusammenhang zwischen $f(t)$ und a_q: $f(g(a_q))$ fällt exponentiell mit wachsendem a_q. Umgekehrt bedeutet dies, daß wir für $a_q < \bar{a}$ mit exponentiell steigender Fehlerrate rechnen müssen!
Im weiteren wird sich noch zeigen, daß es wünschenswert sein kann, auf die optimale Signaturnutzung durch das Attributsollgewicht g_{opt} zu verzichten, damit die somit leichtere Signatur zur Verbesserung des Anfragezeitverhaltens beiträgt. Das ist selbstverständlich nur durch eine Erhöhung der Signaturlänge zu erkaufen.
Aus dem oben Abgeleiteten wird ersichtlich, wie wichtig es für den Signaturentwurf ist, die Analyse einer Stichprobe der der Signatur zugrunde liegenden Daten und Anfragen vorliegen zu haben. Die zwei kritischsten Faktoren in der gesamten Berechnung sind:

1. Das Ermitteln einer möglichst gleichverteilten Abbildung der Wörter auf die Signatur: Sie wird durch den vorhandenen Datenbestand festgelegt, da das eingesetzte Verfahren einer dynamischen Strukturänderung nicht zu folgen vermag. Abgesehen davon ist es i.a. praktisch unmöglich, eine exakte Gleichverteilung zu finden, woraus von vornherein eine gewisse Übergewichtung bestimmter Wörter folgt.

2. Das Festlegen einer mittleren Anzahl $\bar{a}$ der in einer Anfrage spezifizierten Attributwerte: Ist eine weit gestreute Verteilung von a_q zu erwarten, dann steht man vor dem Problem, einen ausgewogenen Kompromiß zwischen System- und Retrievalkosten finden zu müssen. Unterschreitet daraufhin eine Anfrage das angenommene Gewicht, so bezahlt dies der Benutzer mit einer erhöhten Antwortzeit. Dieser Effekt ist wegen der Veränderung der Fehlerwahrscheinlichkeit in 2er- Potenz-Schritten nicht zu unterschätzen. Andererseits steigt die Selektivität in eben diesem Maß, wenn die Anfragesignatur schwerer als das ursprünglich angenommene Gewicht wird. Folglich muß, unter Beachtung des ungünstigsten Falles, für $\bar{a}$ eine vernünftige untere Schranke gewählt werden. Dieselbe Problematik tritt bei der Suche nach der geeigneten durchschnittlichen Anzahl λ Wörter pro Blocksignatur auf, weil diese i.a. ebenfalls variabel ist.

8.4.4.2.3 Codierung für den Substring Match

Superimposed coding auf Zeichenketten angewandt bietet eine elegante Technik für den **Substring-Match**. Hierbei wird ein String durch die Menge seiner unterschiedlichen Substrings der Länge n, die **n-grams**, in Form von überlagerten n-gram-Signaturen repräsentiert. Gängige Werte für n sind 2 und 3. Substrings setzen also notwendigerweise eine Untermenge der Bits der qualifizierenden Strings. Auch hier muß im Anschluß an die Ermittlung der Trefferkandidaten ein pattern match die durch Homonym- und Phantombildung entstandenen false drops entfernen.
In diesem Zusammenhang entsprechen die n-grams den Attributwerten des allgemeinen superimposed codings, wobei die Wertebereiche aller n-grams identisch sind und wir an einer späteren Reidentifizierung der einzelnen n-gram-Positionen nicht interessiert sind. Somit genügt für die Stringkodierung das n-gram-Gewicht $g = 1$. Alles in allem erhält man durch Anwendung der bisherigen Erkenntnisse bei der Stringlänge von l und $\nu = l - n + 1$ n-grams[4] für einen Substring der n-gram-Anzahl ν_Q die Fehlerwahrscheinlichkeit

$$f(\nu_Q) = \left(1 - \left(1 - \frac{1}{L}\right)^{\nu}\right)^{\gamma} \quad \text{mit} \quad \gamma = L \cdot \left(1 - \left(1 - \frac{1}{L}\right)^{\nu_Q}\right). \tag{8.32}$$

Das erwartete Gewicht einer Stringsignatur liegt bei

$$\Gamma = \sum_{i=1}^{L} f(1) = L \cdot \left(1 - \left(1 - \frac{1}{L}\right)^{\nu}\right).$$

[4]Unter realistischer Betrachtung wird man aber von $\nu < l - n + 1$-grams ausgehen, da nach der Überlagerung für diese Berechnung ausschließlich verschiedene n- grams zählen, dieselben n-grams in den meisten Anwendungen aber mehrfach in den Zeichenketten vorkommen.

Setzt man $\Gamma = \frac{1}{2} \cdot L$, dann bekommt man die optimale Signaturlänge

$$L = \frac{1}{1 - 2^{-\frac{1}{\nu}}}$$

mit der zugehörigen Fehlerwahrscheinlichkeit

$$f(\nu_Q) = 2^{-\gamma} = 2^{\frac{1-2^{\nu_Q/\nu}}{1-2^{1/\nu}}} \approx 2^{-\nu_Q}.$$

Normalerweise wird jedoch die optimale Signaturgestaltung den Ansprüchen des Benutzers nicht gerecht werden können, wenn man bedenkt, daß ein Substring mit immerhin $\nu_Q = 6$ noch eine Fehlerwahrscheinlichkeit von über 1 % bedeutet. Daher ist es auch in diesem Fall angebracht, die Signaturlänge über eine akzeptierte Fehlerwahrscheinlichkeit zu bestimmen:

$$\begin{aligned} f(\nu_Q) &\approx \left(1 - \left(1 - \frac{1}{L}\right)^{\nu}\right)^{\gamma} \\ &\approx \left(1 - \left(1 - \frac{\nu}{L}\right)\right)^{\nu_Q} \\ &= \left(\frac{\nu}{L}\right)^{\nu_Q} \end{aligned} \tag{8.33}$$

Die Näherung gilt für $\nu \ll L$ und stellt eine praktikable obere Abschätzung dar. Durch Einsetzen der gewählten Fehlerwahrscheinlichkeit und einige Umformungen liefert dies die zugehörige Signaturlänge

$$L = \nu \cdot \left(\frac{N}{F}\right)^{\frac{1}{\nu_Q}} \tag{8.34}$$

Analog zu den Betrachtungen im vorangegangenen Abschnitt liefern kleine ν und große ν_Q die günstigsten Längen. Zur Vermeidung überladener Signaturen im anschließenden Betrieb, mit überhöhten Fehlerquoten als zwingende Folge, ist es hier genauso angeraten, die Durchschnittswerte eher am schlechteren Ende der Skala anzusiedeln.
In Übereinstimmung mit den vorher gewonnenen Ergebnissen sinkt die Signaturlänge drastisch mit der Zunahme der Suchstringlänge. Dies kann man ebenfalls erreichen, indem man die n-gram-Länge verkürzt, so daß man fälschlicherweise zu der Meinung gelangen könnte, die beste Kodierung ergebe sich für n = 1. An dieser Stelle darf aber nicht außer acht gelassen werden, daß auch das gesamte Signaturpotential, d.h. alle Bitpositionen der Signatur, nutzbar sein muß, was bei der Kardinalität Ω des vorliegenden Alphabets nur für $L \leq \Omega^n$ gilt. Auf diese Weise ist die Fehlerwahrscheinlichkeit in ihrer Güte durch Ω^n begrenzt, weswegen, abhängig von Ω, n nicht zu klein gewählt werden sollte . Abgesehen davon weisen Häufigkeitsanalysen von

n-grams in Texten extreme Ungleichverteilungen auf, woraus direkt folgt, daß bei zunehmender Näherung der Abbildung an eine injektive Funktion ($L \approx \Omega^n$) zwar Homonymfehler immer weiter zurückgehen, die Abbildung dafür aber notwendigerweise bestimmte Bitpositionen übermäßig bevorzugt. Eine ungefähre Gleichverteilung ist hingegen Voraussetzung dieses wahrscheinlichkeitstheoretischen Ansatzes. Andererseits darf n auch nicht zu groß sein, weil das Signaturverfahren erst auf Strings ab Länge n anwendbar ist und infolgedesssen für Suchstrings, die kürzer als n Zeichen sind, alle Datensätze sequentiell durchsucht werden müssen.

10.34 %		35,82 %	
20.68 %	Buchstaben	55.26 %	aller Vorkommen
31.02 %		70.67 %	
1.00 %		19.69 %	
2.00 %	Bigrams	29.69 %	aller Vorkommen
3.00 %		37.57 %	
1.00 %		12.82 %	
2.00 %	Trigrams	19.92 %	aller Vorkommen
3.00 %		25.03 %	

Tabelle 8.4: Empirische Streuung von n-grams in deutschsprachigen Texten

Die oben erwähnte empirische Streuung von n-grams in Texten wurde für die deutsche Sprache bereits von [Bauer & Goos 82] mit dem in Tabelle 8.4 dargestellten Ergebnis untersucht. Derart starke Unregelmäßigkeiten sind äußerst schlecht durch eine Funktion zu glätten, weswegen sich im Umgang mit Zeichenketten der Aufbau von Abbildungstabellen empfiehlt. Zu diesem Zweck werden die zu kodierenden Zeichen in Klassen eingeteilt, so daß jedes Zeichen über seine Klasse eindeutig einem Bit der Signatur zugeordnet werden kann. Anstatt an dieser Stelle gleichgroße Klassen zu bilden, versucht man, Klassen mit gleichgroßer Wahrscheinlichkeit zu schaffen, damit nach Möglichkeit jedes Bit der Signatur mit derselben Wahrscheinlichkeit gesetzt wird. Je besser eine derartige Aufteilung der Zeichen gelingt, desto weniger 1-Bit-Häufungen werden sich in der Signatur bilden, Häufungen, die die angestrebte Neutralität des Systems zu ungunsten spezieller Anfragestrukturen verschieben würden. Dies impliziert, daß bei einer Stichprobenanalyse nur dann Ergebnisse für eine realitätsnahe Kodierung zu erwarten sind, wenn gleichzeitig die morphologische Struktur der Wörter erfaßt wird. Für das Beispiel deutscher Texte heißt das, daß Buchstaben nicht aus dem Kontext ihres Vorkommens herausgerissen betrachtet werden dürfen, denn z.B. tritt c häufig mit h, q mit u auf; daher wäre unter diesem Aspekt n möglichst groß zu wählen.
Praktisch wird das anhand der Buchstabentabelle 8.5 deutlich, die für die Strings der beiden Klassen {'er', 'en'} und {'ee'} unter der Annahme einer unabhängigen

I	Buchstaben	Häufigkeit
0	e	17.3 %
1	ä,j,n,p,x,y	11.8 %
2	r,u	11.8 %
3	c,i,k	11.8 %
4	h,ö,s	11.8 %
5	m,o,t,w	11.8 %
6	d,g,l,q	11.8 %
7	a,b,ü,v,z,f	11.8 %

Tabelle 8.5: Abbildungstabelle für Buchstaben

Verteilung der Buchstaben gleiche Gesamtwahrscheinlichkeiten nahelegt, obwohl nach [Bauer & Goos 82] für Bigrams die Gesamtwahrscheinlichkeit der ersten Klasse mit 3.82 % von der der zweiten mit 0.18 % erheblich differiert. An diesem Punkt erweist sich jedoch nicht die Abhängigkeit von Informationen über die n-gram-Verteilung als problematisch, sondern die Größe der Abbildungstabelle, die schon für kleine n mit Ω^n Einträgen nicht mehr vertretbar ist.
Um die Größe der Abbildungstabelle zu verringern, geht man von der n-gram- zur **t-gram-Kodierung** mit $1 \leq t \leq n$ über. Hierbei wird jedes n-gram in n/t nichtüberlappende t-grams zerlegt, so daß man trotz größerem n mit kleineren t-gram-Tabellen auskommt. Die Transformation wird dann einfach beschrieben durch

$$\begin{aligned} T(\mathrm{n-gram}) &= T(\mathrm{t-gram}_{\frac{n}{t}-1}, \ldots, \mathrm{t-gram}_0) \\ &= \sum_{i=0}^{\frac{n}{t}-1} c^i \cdot I(\mathrm{t-gram}_i). \end{aligned} \tag{8.35}$$

Dabei bildet $I(\mathrm{t-gram})$ jedes t-gram auf einen der c Klassenindices $0 \ldots c-1$ ab. Da die n-grams somit in Zahlen aus dem Intervall $[0, c^{n/t}-1]$ umgesetzt werden, muß die Anzahl c der t-gram-Klassen derart gewählt werden, daß c die größte ganze Zahl ist, für die gilt $c^{n/t} \leq L$.
Die vereinfachende Vorgehensweise durch die Wahl von $t < n$ ist allerdings aufgrund der durch t-gram-Kombinationen möglicherweise entstehenden stark unausgewogenen n-gram-Klassen nicht uneingeschränkt zu empfehlen; es bleibt bloß der Vorteil, im Hinblick auf eine gewünschte Fehlerwahrscheinlichkeit die Signaturlänge L weiter vergrößern zu können. Wird in einer konkreten Anwendung trotzdem ein größerer Wert für n ins Auge gefaßt, dann sollte in jedem Fall zuvor eine Untersuchung über das Wahrscheinlichkeitsgefälle zwischen den entworfenen Klassen und dem realen Vorkommen durchgeführt werden.
Bei der Aufgabe, c möglichst **gleichwahrscheinliche Klassen zu bilden**, handelt es sich um ein *NP*-vollständiges Problem, zu dessen Lösung in [Deppisch 89] folgende

Heuristik vorgeschlagen wird: Zuerst werden alle Symbole nach ihren Häufigkeiten sortiert und durch den Quotienten aus der Summe der Vorkommenshäufigkeiten der noch nicht verteilten Symbole und der Anzahl noch offener Klassen eine Häufigkeitsschranke errechnet. Diese beträgt anfangs $\frac{1}{c}$. Existiert nun ein Symbol, das diese Schranke überschreitet, so bildet es eine eigene Klasse und die Schranke wird neu festgelegt. Diese Verteilung wiederholt sich solange, bis kein Symbol mehr über der aktuellen Schranke liegt. Daraufhin werden die restlichen Symbole in der Reihenfolge absteigender Häufigkeiten auf die verbleibenden Klassen verteilt, indem man jeweils die häufigsten Symbole, die zusammen noch ungefähr die zuletzt fixierte Schranke einhalten, zu einer Klasse zusammenfaßt. Sind alle Symbole Klassen zugeordnet, dann stellt das eine erste Näherung dar, die anschließend durch Austauschen einzelner Symbole noch verbessert werden kann.
Ein weiterer Ansatz zur Verbesserung von Signaturen besteht im sogenannten **Filtern**. Stellt sich durch eine Stichprobe heraus, daß bestimmte Symbole derartig oft auftauchen, daß sie in fast jeder Zeichenkette enthalten sind, so kann man die Selektivität der Signaturen erhöhen, wenn man diese hochfrequenten Symbole in eine Negativmenge aufnimmt, deren Elemente nicht in die Signatur mitaufgenommen werden. Der auf diese Weise konstruierte Filter sollte nicht zu groß sein (laut [Deppisch 89] bis 3 %), weil Signaturen von Anfragen, die diese Symbole enthalten, leichter werden und folglich mehr false drops erzeugen. Dagegen ist das Verhalten der übrigen Anfragen jetzt besser, weil sie von einer größeren Anzahl Homonymfehler befreit werden. Ob es sich lohnt, diese Methode zuzuschalten, hängt demnach sehr davon ab, ob die Symbole der Negativmenge auch wesentliche Bestandteile der Anfragen sind oder nicht.

8.4.4.3 Speicherungsstrukturen

Signaturen an sich bieten bereits eine Beschleunigung der Suche im Vergleich zum Scannen der ursprünglichen Texte. Durch geschickte Wahl der Speicherungsstruktur lassen sich aber noch weitere Effizienzgewinne erzielen. Im folgenden betrachten wir zunächst die einfache sequentielle Organisation und dann einige speziellere Speicherungsstrukturen.

8.4.4.3.1 Sequentielle Signaturen

Abbildung 8.13 zeigt ein Beispiel für die sequentielle Speicherung der Signaturen. Die Signaturen werden dabei getrennt von den eigentlichen Datensätzen in einer eigenen Datei gespeichert, und bei jeder Signatur wird die Adresse des zugehörigen Datensatzes mit abgelegt. Durch diese getrennte Speicherung hat man wesentlich weniger I/O-Operationen als wenn man die Signaturen bei den Datensätzen ablegen würde, denn dann müßten bei der Suche wesentlich mehr Seiten gelesen werden.

	b_1^*	b_2^*	b_3^*	b_4^*	b_5^*	b_6^*	b_7^*	b_8^*	$@R$
S^1	0	0	1	0	1	0	1	1	$@r_1$
S^2	1	0	1	1	1	0	0	0	$@r_2$
S^3	0	1	1	0	0	1	1	0	$@r_3$
S^4	1	0	0	1	0	1	1	1	$@r_4$
S^5	1	1	1	0	0	1	0	0	$@r_5$
S^6	0	1	1	0	0	1	0	1	$@r_6$
S^7	1	0	0	0	1	0	1	0	$@r_7$
S^8	0	0	0	1	1	1	0	1	$@r_8$

Abbildung 8.13: Beispiel für sequentielle Signaturen

Nachstehend werden folgende Bezeichnungen verwendet:

L Länge der Signatur (in Bits)
$size_@$ Länge einer Adresse (in Bytes)
$size_p$ Seitengröße (in Bytes)
$size_r$ Größe eines Datensatzes (in Bytes)
N Anzahl Datensätze in der Datenbasis
M Anzahl Datenseiten

$$M = \left\lceil \frac{N}{\left\lfloor \frac{size_p}{size_r} \right\rfloor} \right\rceil$$

F Anzahl false drops
D Anzahl echter Treffer.

Der Platzbedarf für eine Signatur mit Adresse ist also $\left\lceil \frac{L}{8} \right\rceil + size_@$. Daraus ergibt sich die Anzahl Einträge pro Seite:

$$K = \left\lfloor \frac{size_P}{\left\lceil \frac{L}{8} \right\rceil + size_@} \right\rfloor \tag{8.36}$$

Wir betrachten nun für verschiedene Datenbank-Operationen die **Anzahl Seitenzugriffe**. Für die **Retrieve-Operation** müssen alle Signaturseiten sequentiell gelesen werden:

$$Seq^R = \left\lceil \frac{N}{K} \right\rceil + F + D \tag{8.37}$$

Für die **Insert-Operation** nehmen wir an, daß eine Freispeicherverwaltung in Listenform verwendet wird, so daß nur je ein Lese- und Schreibzugriff für Signatur- und Datenseite notwendig ist:

$$Seq^I = 2 + 2 = 4 \tag{8.38}$$

Bei der **Delete-Operation** sei die Adresse des Datensatzes bekannt. Nun müssen die Signaturseiten sequentiell durchsucht werden, bis der zugehörige Signatureintrag

mit dieser Adresse gefunden ist. Diese Seite muß geändert zurückgeschrieben werden, ebenso muß die Datenseite geändert werden.

$$Seq^D = \left\lceil \frac{N}{2 \cdot K} \right\rceil + 1 + 2 = \left\lceil \frac{N}{2 \cdot K} \right\rceil + 3 \tag{8.39}$$

Der gesamte **Speicherplatzbedarf** (in Seiten) für die Datenbank mit sequentiellen Signaturen ist

$$Seq^S = \left\lceil \frac{N}{K} \right\rceil + M. \tag{8.40}$$

8.4.4.3.2 Bitscheibenorganisation

	b_1^*	b_2^*	b_3^*	b_4^*	b_5^*	b_6^*	b_7^*	b_8^*	$@R$
S^1	0	0	1	0	1	0	1	1	$@r_1$
S^2	1	0	1	1	1	0	0	0	$@r_2$
S^3	0	1	1	0	0	1	1	0	$@r_3$
S^4	1	0	0	1	0	1	1	1	$@r_4$
S^5	1	1	1	0	0	1	0	0	$@r_5$
S^6	0	1	1	0	0	1	0	1	$@r_6$
S^7	1	0	0	0	1	0	1	0	$@r_7$
S^8	0	0	0	1	1	1	0	1	$@r_8$

Abbildung 8.14: Beispiel zur Bitscheibenorganisation

Eine wesentlich effizientere Organisationsform für Signaturen ist die Bitscheibenorganisation, wie sie beispielhaft in Abbildung 8.14 dargestellt ist. Hierbei wird jede Bitposition („Bitscheibe“) der Signaturen in einer eigenen Seite gespeichert. Die zugehörigen Datensatzadressen werden getrennt hiervon in einem gesonderten Bereich abgelegt.
Der wesentliche Vorteil dieser Organisationsform liegt darin, daß man beim Retrieval nur diejenigen Bitscheiben zu lesen braucht, für die in der Anfragesignatur eine 1 gesetzt ist. Durch UND-Verknüpfung dieser Bitscheiben erhält man die Ergebnisbitliste, zu der man dann die zugehörigen Datensatzadressen holen muß. Diese Vorgehensweise soll hier anhand eines Beispiels verdeutlicht werden. Wir nehmen an, die Anfrage sei: $S^i \supseteq_S <10100000>$. Damit erhalten wir zu den Signaturen aus Abbildung 8.14 folgende Ergebnisbitliste:

$$b_r^* = b_1^* \wedge b_3^*$$

$$= \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \wedge \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Allgemein berechnet man die Ergebnisbitliste nach der Formel:

$$b_r^* = \bigwedge_{j=q_1}^{q_{\gamma(S_Q)}} b_j, \quad q_i \in \{q | (1 \leq q \leq L) \wedge b_q^Q = 1\} \tag{8.41}$$

Die Menge der Adressen der Trefferkandidaten ist dann $R = \{i | b_r^i = 1\}$.
Bei disjunktive Anfragen der Form $S(Q_1) \vee S(Q_2) \vee \ldots S(Q_T)$ muß im Prinzip eine getrennte Prozessierung für die einzelnen Minterme $S(Q_i)$ durchgeführt werden, nur bei übereinstimmenden 1-Bits in den $S(Q_i)$ sind Einsparungen möglich.
Zur Berechnung der **Anzahl Seitenzugriffe** für Datenbank-Operationen gehen wir von einem Speicherbedarf für eine Bitscheibe in Höhe von $\lceil N/(8 \cdot size_p) \rceil$ aus. Bei der **Retrieve-Operation** müssen zunächst die angesprochenen Bitscheiben gelesen werden, dann die Einträge aus der Zuordnungstabelle entsprechend der Ergebnisbitliste und schließlich die zugehörigen Datenseiten. Nun bezeichne $\gamma(Q)$ das Anfragegewicht eines Minterms und T die Anzahl Minterme in der Anfrage. Damit ergibt sich die Anzahl Seitenzugriffe zu

$$BS^R = T \cdot \left(\gamma(Q) \left\lceil \frac{N}{8 \cdot size_p} \right\rceil + Z + F + D \right) \tag{8.42}$$

Hierbei bezeichnet Z die Anzahl Seitenzugriffe auf die Zuordnungstabelle. Die Gesamtzahl Seiten der Zuordnungstabelle ist

$$R = \left\lceil \frac{N}{\left\lfloor \frac{size_p}{size_{@}} \right\rfloor} \right\rceil . \tag{8.43}$$

Damit läßt sich dann der Erwartungswert von Z berechnen zu:

$$Z = R \cdot \left(1 - \left(1 - \frac{1}{R} \right)^{(F+D)} \right) \tag{8.44}$$

Der prinzipieller Nachteil von Bitscheiben bei Retrieve-Operationen besteht darin, daß sie bei hohen Anfragegewichten ineffizient werden. Als Abhilfe kann man die Anfrage

nur für einen Teil der gesetzten Bits auswerten, woraus sich natürlich eine höhere Anzahl false drops ergibt. Es existiert aber ein Grenzgewicht für Anfragen, ab dem die durch false drops verursachten I/O-Operationen geringer sind als die Operationen für die Auswertung weiterer Bitscheiben.
Für die **Insert-Operation** nehmen wir an, daß die Bitscheibenblöcke vorher mit 0 initialisiert wurden. Bezeichne $\gamma(S)$ das Signaturgewicht des einzufügenden Datensatzes, dann müssen $\gamma(S)$ Bitscheibenseiten, der Eintrag in der Zuordnungstabelle und die Datenseite geändert werden:

$$BS^I = 2 \cdot \gamma(S) + 2 + 2. \tag{8.45}$$

Bei der **Delete-Operation** wird der Eintrag (mit bekannter Satzadresse) über die Signatur gesucht. Dann müssen für alle 1-Bits in der Satzsignatur die Bitscheibenseiten geändert werden und zusätzlich die Zuordnungstabelle und die Datenseite:

$$BS^D = \gamma \cdot \left(\left\lceil \frac{N}{8 \cdot size_p} \right\rceil + 1 \right) + Z + 2. \tag{8.46}$$

Der **Speicherplatzbedarf** (in Seiten) bei Bitscheibenorganisation beträgt

$$BS^S = L \cdot \left\lceil \frac{N}{8 \cdot size_p} \right\rceil + \left\lceil N \cdot \frac{size_@}{size_p} \right\rceil + M. \tag{8.47}$$

8.4.4.3.3 Zweistufiges Signaturverfahren

Das zweistufige Signaturverfahren versucht die Vorteile von Bitscheiben- und sequentieller Organisation zu kombinieren. Hierbei werden für jeden Datensatz zwei Signaturen mit voneinander unabhängigen Signaturfunktionen berechnet. Die erste Signatur wird wie bei sequentieller Organisation als Bitstring in Seiten gespeichert, wobei die Signatur-Seiten aber in Segmente unterteilt sind. Entsprechend dieser Segment-Einteilung werden die zweiten Signaturen jedes Datensatzes nun segmentweise überlagert zur sogenannten Segmentsignatur. Diese werden in Bitscheibenorganisation verwaltet.
Bei der Anfrageprozessierung werden mit den beiden Signaturfunktionen zwei Anfragesignaturen berechnet. Mit der zweiten Signatur werden nun über die Bitscheiben die zu durchsuchenden Segmente bestimmt. Die Signaturseiten dieser Segmente werden dann sequentiell durchsucht.

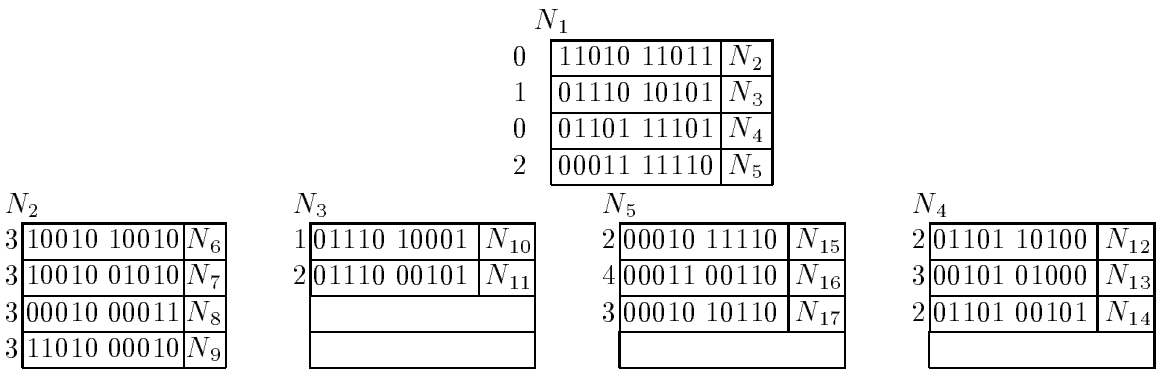


Abbildung 8.15: Beispiel für einen S-Baum

8.4.4.3.4 S-Baum

So wie der B*-Baum für numerische Daten und Zeichenketten sowohl effiziente Such- wie auch Änderungsoperationen erlaubt, so soll der S-Baum dies für Signaturen ermöglichen. Abbildung 8.15 zeigt ein Beispiel hierzu. Jeder Knoten besteht aus einer Menge von Einträgen der Form (Signatur, Adresse), wobei die Adresse entweder auf einen untergeordneten Knoten des S-Baums oder bei Blattknoten auf einen Datensatz zeigt. Bei dem Wurzelknoten und den inneren Knoten des Baumes entsteht die Signatur eines einzelnen Eintrages durch die Überlagerung der Signaturen des Knotens, auf den die Adresse verweist. (Die in Abb. 8.15 vor den Knoteneinträgen stehenden Ziffern sind hier zunächst ohne Bedeutung.)
Wir verwenden nachstehend folgende Notationen:

- K Knotenkapazität = maximale Anzahl Einträge der Form (Signatur, Adresse),
- k minimale Anzahl Einträge in einem Knoten
- h Höhe des Baums
- R Menge der Datensätze
- E Signatur-Eintrag der Form (Signatur, Adresse)
- $E.S$ Signatur
- $E.p$ Adresse
- N_p^i Menge der Einträge des Knotens auf der Stufe i mit der Adresse p

Ein Signaturbaum ist ein S-Baum vom Typ (K, k, h) mit $K, k, h \in I\!N_0$, wenn er die folgenden Eigenschaften besitzt:

1. Jeder Nicht-Blattknoten der Stufe i enthält Signaturen für Knoten der Stufe $i-1$ gemäß der Gleichung

$$N_p^i = \{E_j | 1 \leq j \leq K \wedge E_j.S = \bigvee_{E \in N^{i-1}_{E_j.P}} E.S\} \quad \text{für } 1 < i \leq h. \tag{8.48}$$

2. Jedes Blatt enthält Satzsignaturen gemäß der Gleichung

$$N_p^1 = \{E_j | 1 \leq j \leq K \wedge E_j.S = S(r_j) \wedge\ E_j.p = @r_j \wedge r_j \in R\}\,. \tag{8.49}$$

3. Jeder Knoten hat höchstens K Einträge: $(\forall p)(|N_p| \leq K)$.
4. Jeder Pfad von der Wurzel zu einem Blatt hat dieselbe Länge h.
5. Die Wurzel enthält mindestens 2 Einträge: $|N_w^h| \geq 2$.
6. Jeder Nicht-Wurzelknoten hat mindestens k Einträge: $(\forall p \neq w)(|N_p| \geq k \leq \lfloor \frac{K}{2} \rfloor)$.

Da jeder Knoten eines S-Baums zwischen k und K Einträge enthält, gilt für dessen Höhe h:

$$h_{min} = \lceil \log_K N \rceil \leq h \leq \lfloor \log_k N \rfloor = h_{max} \tag{8.50}$$

Man kann ferner zeigen [Deppisch 89], daß die maximale Anzahl der Knoten im S-Baum

$$1 + 2 \left\lfloor \sum_{i=1}^{h_{max}-1} \frac{N}{k^i} \right\rfloor + h_{max} - 1 = h_{max} + 2 \left\lfloor \left(\frac{\left(\frac{1}{k}\right)^{h_{max}} - 1}{\frac{1}{k} - 1} - 1 \right) N \right\rfloor \tag{8.51}$$

ist und das für den Speicherplatzbedarf S^S (in Seiten) gilt:

$$\left\lceil \left(\frac{\left(\frac{1}{K}\right)^{h_{min}} - 1}{\frac{1}{K} - 1} - 1 \right) N \right\rceil \leq S^S \leq h_{max} + 2 \left\lfloor \left(\frac{\left(\frac{1}{k}\right)^{h_{max}} - 1}{\frac{1}{k} - 1} - 1 \right) N \right\rfloor . \tag{8.52}$$

Wir betrachten nun die Operationen auf dem S-Baum, und zwar zunächst die **Retrieve-Operation**. Hierzu muß der Baum rekursiv durchsucht werden, wobei im Gegensatz etwa zum B*-Baum mehrere Teilbäume Treffer enthalten können. Die nachstehend dargestellte Prozedur RETRIEVE erhält als Argumente die Anfragesignatur und die Seitennummer eines Knotens des S-Baums. Die gefundenen Satzadressen werden mit der Prozedur DELIVER übergeben.

```
procedure RETRIEVE(S_Q: Signatur; p: Seitennummer);
with N_p^i do
    for each non-empty E in N_p^i do
        if E.S ⊇_S S_Q then do
          if i > 1
            then RETRIEVE (S_Q, E.p)
            else DELIVER (E.p)
        end;
    end;
end;
end;
```

Von den anderen Operationen wollen wir hier nur die **Insert-Operation** betrachten. Hierzu muß ein geeigneter Knoten gesucht werden, in den die neue Signatur eingefügt wird. Da man prinzipiell eine Signatur in jeden beliebigen Knoten einfügen kann, muß man hierfür ein Entscheidungskriterium vorgeben. Optimal wäre natürlich, denjenigen Knoten auszuwählen, der die Anzahl der zu erwartenden I/O-Operationen (insbesondere beim Retrieval) minimiert. Da dies aber praktisch nicht möglich ist, liegt es nahe, denjenigen Knoten auszuwählen, der den geringsten Gewichtszuwachs in den Vaterknoten verursacht. Als Beispiel hierzu ist vor den Einträgen in Abbildung 8.15 der Gewichtszuwachs angegeben, wenn in dem darunterliegenden Knoten die Signatur $< 0100011001 >$ eingefügt würde. Um nun einen Knoten mit minimalem Gewichtszuwachs zu finden, wird eine Breitensuche gemäß der nachstehend dargestellten Funktion CHOOSE durchgeführt. Diese erhält als Eingabeparameter die einzufügende Signatur und eine Menge von zu durchsuchenden Seitennummern (zu Beginn wird der Wurzelknoten übergeben); als Ergebnis liefert CHOOSE die Nummer der Seite, in die die Signatur einzufügen ist.

```
function CHOOSE(S: Signatur,P: Menge Seitennummern): Seitennummer;
if i > 1
then do
    ε_min := maxint;
    for each p ∈ P do
        with N_p^i do
            for each non empty E(j) ∈ N_p^i do
                e := ε(E(j).S, S);
                if e < ε_min
                then do
                        P_min := {p};
                        ε_min := e;
                        Linkson (Tree, p, j, E(j).p);
                end
                else do
                        if e = ε_min
                        then do
                                P_min := P_min ∪ p;
                                Linkson (Tree, p, j, E(j).p);
                        end;
                end;
            end;
        end;
    end;
    CHOOSE := CHOOSE(S, P_min)
end
```

else CHOOSE := first $p \in P$;
end;
Beim Insert kann es zu einem Überlauf des Knotens kommen, in den eingefügt werden soll. Dann muß dieser Knoten gesplittet werden. Man kann zeigen, daß die Bestimmung eines optimalen Splits ein NP-vollständiges Problem ist [Deppisch 89]. Deswegen wird stattdessen ein heuristischer Algorithmus nach dem Greedy-Prinzip angewendet. Dabei werden zunächst zwei Saatsignaturen als Initialisierung der beiden neuen Knoten ausgewählt, und dann die übrigen Signaturen auf die beiden Knoten verteilt.
Die Initialisierung des Splits übernimmt die Prozedur INITSPLIT:
procedure INITSPLIT; (*lokal*)
$p1$:= newpage :
$p2$:= newpage :
$N_p^i := N_p^i \cup E$; (*virtuelles Einfügen*)
$seed_1, seed_2$:= PICKSEEDS(p);
$N_{p1}^i := N_{p1}^i.E(seed_1)$;
$N_{p2}^i := N_{p2}^i.E(seed_2)$;
$S_{seed_1} := N_p^i.E(seed_1).S$;
$S_{seed_2} := N_p^i.E(seed_2).S$;
$N_p^i := N_p^i \backslash N_{p1}^i.E(seed_1) \backslash N_{p2}^i.E(seed_2)$;
end;
Die Funktion PICKSEEDS erhält als Parameter die Nummer der (übervollen) Seite, aus der die beiden Saatsignaturen wie folgt ausgewählt werden: $seed_1$ ist die Signatur S_1 mit dem maximalen Gewicht $\gamma(S)$, und $seed_2$ ist die zugehörige Signatur S_2 mit der maximalen Distanz $\epsilon(S_2, S_1)$ (Anzahl unterschiedlicher Bits) zu S_1.
function PICKSEEDS$_{lin}$ (p: Seitennummer): ($seed_1, seed_2$: Eintragsnummer);
with N_p^i do
    $seed_1$:= first$j \in \{j | \max_j(\gamma(E(j).S))\}$;
    $seed_2$:= first$j \in \{j | \max_j(\epsilon(E(seed_1).S, E(j).S))\}$;
    PICKSEEDS := $(seed_1, seed_2)$;
end;
end;
Anschließend müssen die übrigen Signaturen auf die beiden neuen Knoten verteilt werden. Als Strategie wird hierbei versucht, eine ausgeglichene Zuordnung zur Knotensignatur (nach Gewichtszuwachs) zu erreichen. Hierzu wird bei jedem Durchgang abwechselnd für jeden Knoten die Signatur mit der höchsten Priorität eingefügt, dann wird die betroffene Knotensignatur aktualisiert und die Prioritäten neu ausgewertet.
Die zugehörige Funktion SPLIT erhält als Parameter den neu einzufügenden Eintrag und die Seitennummer des übergelaufenen Knotens, zurückgeliefert werden die Einträge für die beiden neu angelegten Knoten.
function SPLIT(E: Eintrag, p: Seitennummer): (Eintrag,Eintrag);
INITSPLIT;

with N_p^i do
  for $l := 1$ to $\left\lceil \frac{K-1}{2} \right\rceil$ do
    $j :=$ first$j \in \{j | \max_j(\epsilon(S(N_{p2}^i), E(j).S) - \epsilon(S(N_{p1}^i), E(j).S))\}$;
    $N_{p1}^i := N_{p1}^i \cup E(j)$;
    $N_p^i := N_p^i \backslash E(j)$;
    if $|N| > 0$
    then do
      $j :=$ first$j \in \{j | \max_j(\epsilon(S(N_{p1}^i), E(j).S) - \epsilon(S(N_{p2}^i), E(j).S))\}$;
      $N_{p2}^i := N_{p2}^i \cup E(j)$;
      $N_p^i := N_p^i \backslash E(j)$;
    end;
  end;
  $N_p^i := N_{p2}^i$;
  SPLIT $:= ((p, S(N_p^i)), (p1, S(N_{p1}^i)))$;
end;
end;

8.4.4.3.5 Quick Filter

Eine ähnlich effiziente Organisationsform wie der S-Baum ist der in [Rabitti & Zezula 90] vorgestellte Quick Filter. Dieser stellt eine Kombination von Signaturen mit Hashing dar. Die Signaturen sind hier wieder in Seiten organisiert, wobei die Zuordnung der Signaturen zu den Seiten über Hashing erfolgt. Hierfür gibt es n Primärseiten, wobei
jede Primärseite 0 oder mehr Überlaufseiten haben kann, die mit dieser verkettet sind.
Als Hash-Funktion wird **lineares Hashing** verwendet. Dabei bildet die lineare Hash-Funktion g die Schlüssel auf den Adreßraum $(0, 1, \ldots, n-1)$ ab. Für die Anzahl Stufen h der Signaturdatei (mit $h \in I\!N$) gilt dabei $2^{h-1} < n \leq 2^h$. Die Hash-Funktion $g(S, h, n)$ für eine Signaturen $S =< b_1, \ldots, b_L >$ ist dabei wie folgt definiert:

$$g(S, h, n) = \begin{cases} \sum_{r=0}^{h-1} b_{L-r} 2^r, & \text{falls } \sum_{r=0}^{h-1} b_{L-r} 2^r < n \\ \sum_{r=0}^{h-2} b_{L-r} 2^r, & \text{sonst} \end{cases} \tag{8.53}$$

Es werden also abhängig von n die rechten $h-1$ bzw. $h-2$ Bits der Signatur als Binärzahl interpretiert.
Zu Beginn sei $h = 0$, $n = 1$ und $g(S, 0, 1) = 0$. Beim **Einfügen eines neuen Schlüssels** K wird zunächst die Seitennummer $p = g(K, h, n)$ berechnet. Wenn diese Seite noch nicht voll ist, wird der Schlüssel hier eingefügt, anderenfalls in einer Überlaufseite zu p. Bei Überlauf wird zusätzlich der Adreßraum von n auf $n+1$ vergrößert.

Zum **Vergrößern des Adreßraums** existiert ein Zeiger SP auf die nächste zu splittende Seite. Es wird nun eine neue Primärseite n angelegt und der Inhalt der Seite SP und der zugehörigen Überlaufseiten durch die neue Hashfunktion auf die Seiten SP und n verteilt. Anschließend wird n um 1 erhöht; zusätzlich wird h um 1 erhöht, wenn die Seite 0 gesplittet wurde. Schließlich wird der neue Splitzeiger nach der Formel $SP := (SP + 1) \bmod 2^{h-1}$ berechnet.
Abbildung 8.16 zeigt diese Vorgehensweise beim Einfügen von sechs Signaturen in Seiten mit der Kapazität 2.

	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
S_1	0	0	0	1	1	1	1	0
S_2	1	1	0	1	0	0	0	1
S_3	0	0	1	1	1	1	0	0
S_4	1	1	0	0	0	0	1	1
S_5	0	0	1	1	0	1	1	0
S_6	1	1	0	0	1	0	0	1

Step 0.	**P0**: leer	$SP = 0, h = 0, n = 1$
Step 1.	**P0**: S1	$SP = 0, h = 0, n = 1$
Step 2.	**P0**: S1 S2	$SP = 0, h = 0, n = 1$
Step 3.	**P0**: S1 S3 **P1**: S2	$SP = 0, h = 1, n = 2$
Step 4.	**P0**: S1 S3 **P1**: S2 S4	$SP = 0, h = 1, n = 2$
Step 5.	**P0**: S3 **P1**: S2 S4 **P2**: S1 S5	$SP = 1, h = 2, n = 3$
Step 6.	**P0**: S3 **P1**: S2 S6 **P2**: S1 S5 **P3**: S4	$SP = 0, h = 2, n = 4$

P0: 00111100	**P1**: 11010001 11001001
P2: 00011110 00110110	**P3**: 11000011

Abbildung 8.16: Einfügen von Signaturen beim Quick Filter

Beim **Retrieval** müssen die zu durchsuchenden Signaturseiten aus der Anfragesignatur bestimmt werden. Dabei hängt die Anzahl der zu lesenden Seiten vom Gewicht der Anfragesignatur Q ab, genauer von der Anzahl der Einsen in den letzten h Bits: enthält Q j Einsen im h-Bit-Suffix $h(Q)$, dann müssen höchstens 2^{h-j} Primärseiten und die zugehörigen Überlaufseiten gelesen werden. Der zugehörige Algorithmus sieht dann wie folgt aus:

1. $P := g(Q, h, n)$
2. if $h(Q) \cap P = h(Q)$ then {Signaturseite P lesen}
3. $P := P + 1$
4. if $P < n$ then goto 2
5. end.

	Profil			
Struktur	Retrieve	Insert	Delete	Speicher
sequentiell	selten	dominant!	selten	dominant
Bitscheiben	dominant	wenig	wenig	wenig
zweistufig	dominant	wenig	wenig	irrelevant
Quick Filter, S-Baum	wenig	viel	viel	wenig

Tabelle 8.6: Anforderungsprofile der Speicherungsstrukturen

8.4.4.4 Vergleich der Speicherungsstrukturen

In Tabelle 8.6 sind die vorgestellten Speicherungsstrukturen noch einmal vergleichend gegenübergestellt. Ausgehend von dem Anforderungsprofil der geplanten Anwendung, d.h. der Häufigkeit der einzelnen Operationen, kann man daraus die geeignete Speicherungsstruktur ersehen.

8.4.4.5 Ranking mit Signaturen

Bei den Retrieve-Operationen zu den verschiedenen Speicherungsstrukturen wurden immer boolesche Anfragen angenommen. Uns interessiert natürlich mehr, wie man Ranking mit Signaturen durchführen kann. In [Schmidt & Pfeifer 93] wird diese Problemstellung erstmals systematisch untersucht, und zwar für den Fall des S-Baums. Zum Messen der Anzahl False Drops beim Ranking vergleichen wir mit einem Ranking auf der Basis von invertierten Listen (hier: das SMART-System). Dabei haben wir das Problem, daß aufgrund der binären Dokumentindexierung mehrere Dokumente im gleichen Rang landen können. Abbildung 8.17 illustriert die hierzu verwendete Meßmethode: Wir wollen die ersten 10 Dokumente finden, wobei aber nicht notwendigerweise eine Ranggrenze nach 10 Dokumenten vorliegt. Daher suchen wir zunächst alle Dokumente aus den vorigen Rängen und dann noch weitere Dokumente aus dem letzten betrachteten Rang, bis wir insgesamt 10 Dokumente gefunden haben.
Nachstehend werden folgende Notationen zugrundegelegt:

- L Länge der Signatur in Bits.
- g Termgewicht; Anzahl der pro Term gesetzten, unterschiedlichen Bitpositionen.
- λ Anzahl der überlagerten Termsignaturen
- q_k Frage, repräsentiert durch Frageterme $q_{k,i}$, $i = 1, \ldots, n_k$
- n_k Anzahl Frageterme der Frage q_k
- d_m Dokument, repräsentiert durch Indexierungsterme $d_{m,i}$
- $S(\cdot)$ Funktion, die für einen einzelnen Term bzw. ein/e Dokument/Frage eine Signatur berechnet:
  $S(d_m) = S(d_{m,1}) \vee S(d_{m,2}) \vee, \ldots, S(d_{m,n})$, wenn n die Anzahl der Indexierungsterme des m-ten Dokuments ist.

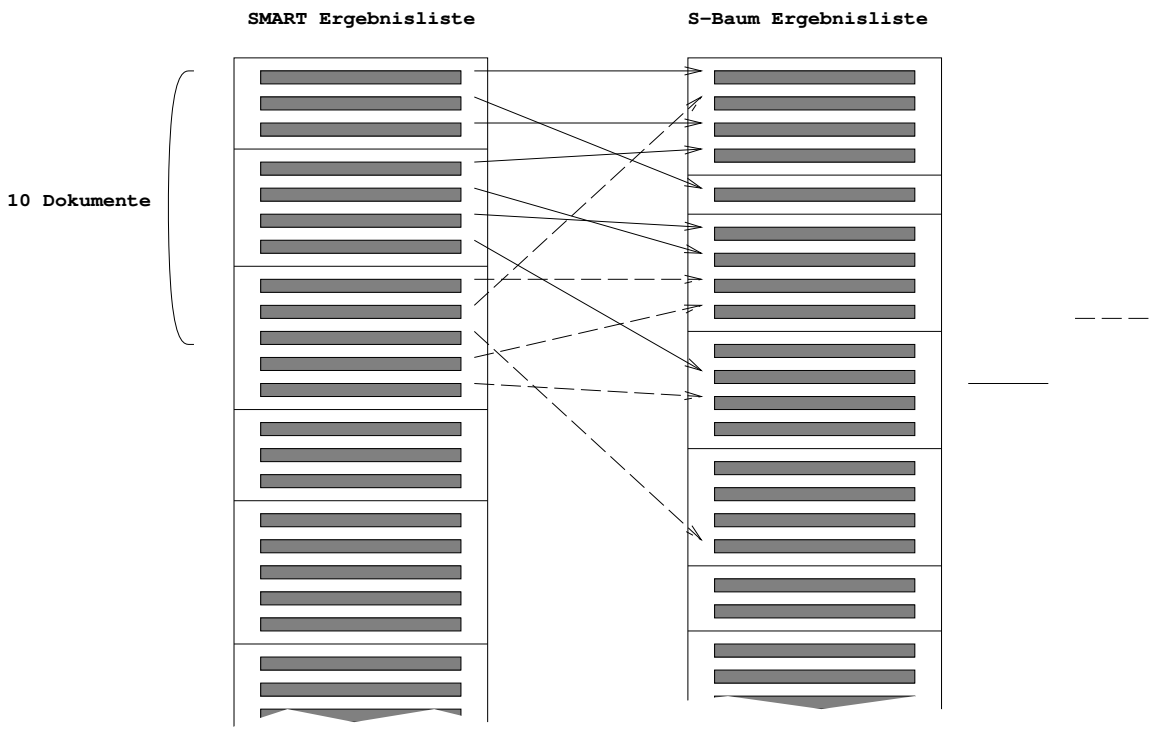


Abbildung 8.17: Meßmethode

Für das Ranking wurden zwei verschiedene Retrievalfunktionen für den Vergleich von Frage- und Dokumentsignatur verwendet, die beide den Coordination Level Match annähern sollen. Die erste Funktion zählt nur die Anzahl übereinstimmende Bits, die zweite prüft dagegen termweise ab, ob alle Bits des Terms in der Dokumentsignatur vorkommen, und zählt dann die Anzahl der so gefundenen Terms

8.4.4.5.1 Retrieval mit der Anzahl übereinstimmender Bits

Die Retrievalfunktion lautet hier:

$$\varrho_{Best\text{-}Bit}(q_k, d_m) := |S(q_k) \wedge S(d_m)| \tag{8.54}$$

Um die Anzahl der zu erwartenden False Drops abzuschätzen, berechnen wir zunächst die Wahrscheinlichkeit, daß ein bestimmtes Bit nach der Überlagerung von λ Signaturen der Länge L mit dem Gewicht g gesetzt ist:

$$p(L, g, \lambda) = 1 - (1 - \frac{g}{L})^{\lambda}.$$

Die Wahrscheinlichkeit, daß i bestimmte Bits in der Dokumentsignatur gesetzt sind, ist dann $p(L, g, \lambda)^i$.
Die Anzahl der von einer Frage mit n_k Terms gesetzten Bits nähern wir mit $n_k \cdot g$ an. Die Retrievalfunktion zählt nun, wieviele dieser $n_k \cdot g$ Bits in der jeweiligen Dokumentsignatur gesetzt sind. Die Wahrscheinlichkeit $f_{eB}(L, g, n_k, i)$, daß **genau** i dieser Bits zufällig gesetzt sind, ist

$$f_{eB}(L, g, n_k, i) = \binom{n_k \cdot g}{i} \cdot p(L, g, \lambda)^i \cdot (1 - p(L, g, \lambda))^{n_k \cdot g - i}$$

Für eine bestimmten Rang sei j (mit $0 \leq j \leq n_k \cdot g$) die Anzahl der zwischen Frage- und Dokumentsignatur übereinstimmenden Bits. Die Wahrscheinlichkeit $f_B(L, g, n_k, j)$, daß **mindestens** j Bits zwischen Frage- und Dokumentsignatur zufällig übereinstimmen, ist dann unsere gesuchte False-Drops-Wahrscheinlichkeit

$$f_B(L, g, n_k, j) = \sum_{j \leq i \leq n_k \cdot g} f_{eB}(L, g, n_k, i). \tag{8.55}$$

Da wir nicht wissen, bei wievielen Bits Abweichung wir die 10 gesuchten Dokumente finden werden, betrachten wir alle möglichen Werte $f_B(L, g, n_k, j)$ mit $0 \leq j \leq n_k \cdot g$ und bilden daraus den Mittelwert. Abbildung 8.18 zeigt am Beispiel der CACM-Kollektion (3200 Dokumente, $L = 509$) für $n_k = 10$ und $g = 1, \ldots, 12$ die sich ergebenden False Drops. Offensichtlich ist für $g = 1$ die False-Drops-Rate am niedrigsten. Dies wird durch die tatsächlich sich ergebenden False Drops, die in Abbildung 8.19 dargestellt sind, bestätigt.

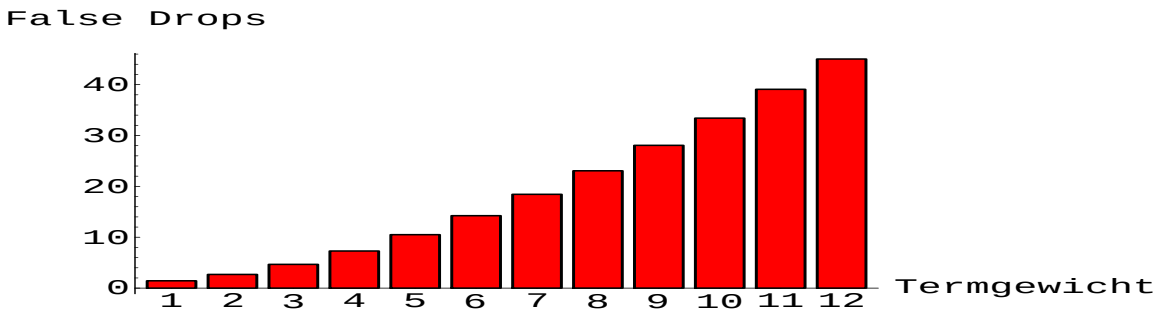


Abbildung 8.18: Theoretische False-Drops-Rate bei $\varrho_{Best\text{-}Bit}$

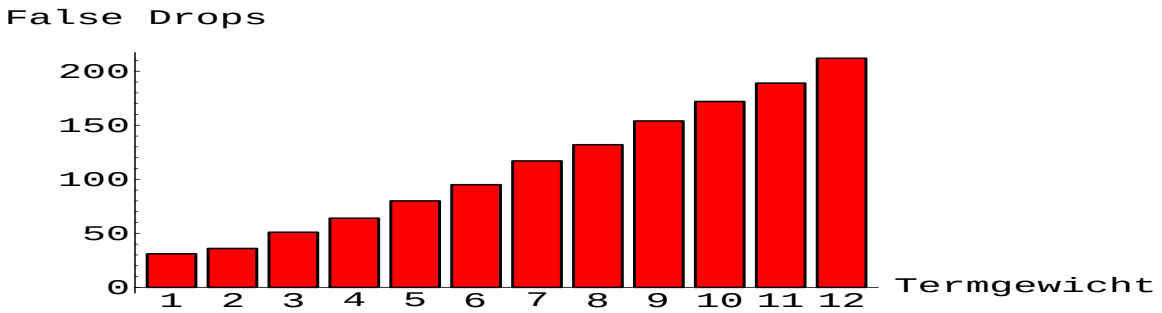


Abbildung 8.19: Tatsächliche False-Drops-Rate bei $\varrho_{Best\text{-}Bit}$

8.4.4.5.2 Retrieval mit der Anzahl übereinstimmender Terms

Die Retrievalfunktion $\varrho_{Best\text{-}Term}$ zählt hier, wieviele Fragetermsignaturen in der Dokumentsignatur gefunden werden:

$$\varrho_{Best\text{-}Term}(q_k, d_m) := \sum_{i=1}^{n_k} (S(q_{k,i}) \subseteq S(d_m)) \tag{8.56}$$

Die Wahrscheinlichkeit, daß ein bestimmter Term (der g Bits setzt) gefunden wird, ist hier $p(L, g, \lambda)^g$. Daraus ergibt sich die Wahrscheinlichkeit $f_{eT}(L, g, n_k, i)$, daß **genau** i der n_k Terms der Anfrage in einer zufälligen Signatur gefunden werden, zu

$$f_{eT}(L, g, n_k, i) = \binom{n_k}{i} p(L, g, \lambda)^{g \cdot i} \cdot (1 - p(L, g, \lambda)^{g \cdot (n_k - i)}$$

Unsere gesuchte False-Drops-Wahrscheinlichkeit ist die Wahrscheinlichkeit $f_T(L, g, n_k, j)$, daß **mindestens** j Terms der Anfrage (mit $0 \leq j \leq n_k$) in der Dokumentsignatur zufällig gefunden werden:

$$f_T(L, g, n_k, j) = \sum_{i=j}^{n_k} f_{eT}(L, g, n_k, i) \tag{8.57}$$

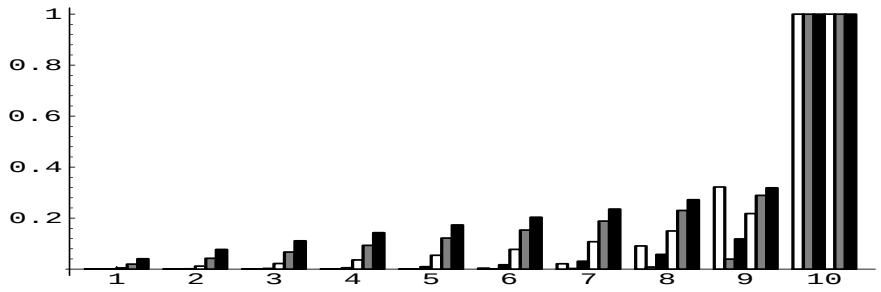


Abbildung 8.20: Theoretische False-Drops-Wahrscheinlichkeit bei $\varrho_{Best\text{-}Term}$

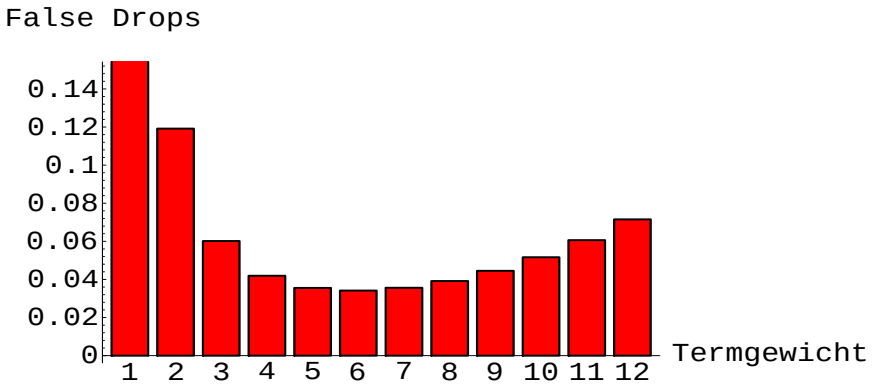


Abbildung 8.21: Mittlere theoretische False-Drops-Wahrscheinlichkeit bei $\varrho_{Best\text{-}Term}$

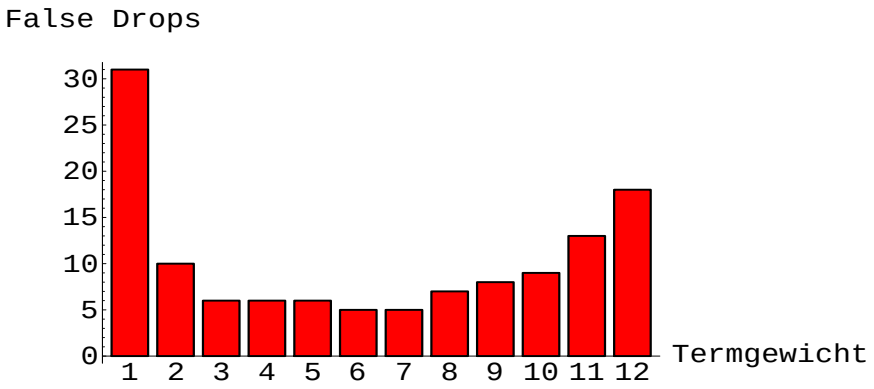


Abbildung 8.22: Tatsächliche False-Drops-Rate bei $\varrho_{Best\text{-}Term}$

Abbildung 8.20 zeigt die Werte dieser Funktion für $n_k = 10$, $g = 1, 10, 20, 30, 40, 50$ und die Ränge $(= n_k - j)$ $1 \ldots, 10$, wobei man erkennt, daß die minimale False-drops-Rate sich wohl für g zwischen 1 und 20 ergibt. Um dies näher zu untersuchen, betrachten wir wieder die False-Drops-Wahrscheinlichkeit für alle möglichen Werte j mit $0 \leq j \leq n_k$ und bilden daraus den Mittelwert. Dieser ist in Abbildung 8.21 für $g = 1, \ldots, 12$ dargestellt. Die tatsächlichen False-Drops-Raten zeigt Abbildung 8.22, die auch das theoretische Minimum bei $g = 6$ bestätigt.

8.4.4.5.3 Besuchte Knoten des S-Baums

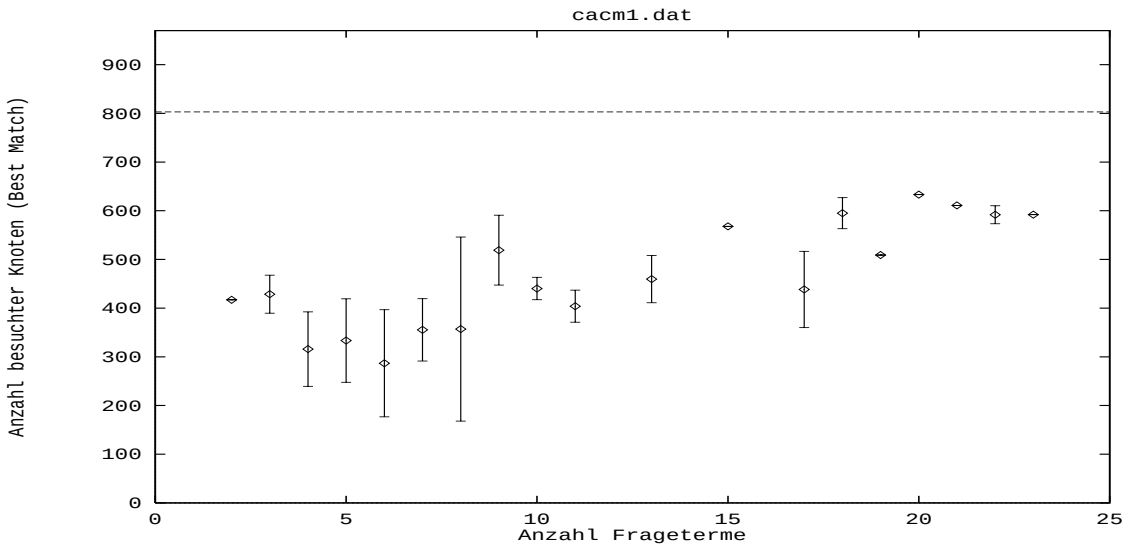


Abbildung 8.23: Besuchte Knoten bei $\varrho_{Best\text{-}Bit}$, $g = 1$

Ziel des S-Baums ist es ja, beim Retrieval die Anzahl der Vergleiche zwischen Frage- und Dokumentsignaturen zu verringern. Die Abbildungen 8.23–8.25 zeigen die Anzahl der besuchten Knoten bei den verschiedenen hier betrachteten Retrievalfunktionen. Es wird deutlich, daß stets ein hoher Prozentsatz der Knoten untersucht werden muß. Somit läßt sich feststellen, daß der S-Baum für Best-Match-Anfragen wenig geeignet ist. Diese Aussage gilt wohl in ähnlicher Form auch für den Quick Filter. Somit scheinen Signaturen generell wenig geeignet für Retrieval mit Ranking.

8.4.5 PAT-Bäume

Der PAT-Baum ist eine Datenstruktur zur effizienten Textsuche in großen Datenbeständen. Insbesondere kann auch die Suche in strukturierten Dokumenten effektiv

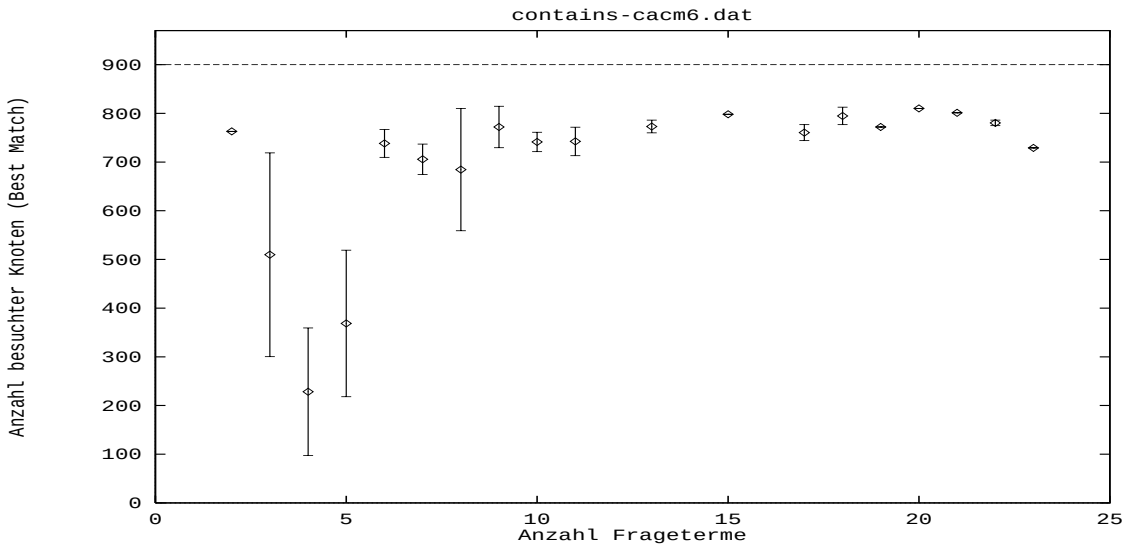


Abbildung 8.24: Besuchte Knoten bei $\varrho_{Best\text{-}Term}$, $g = 6$

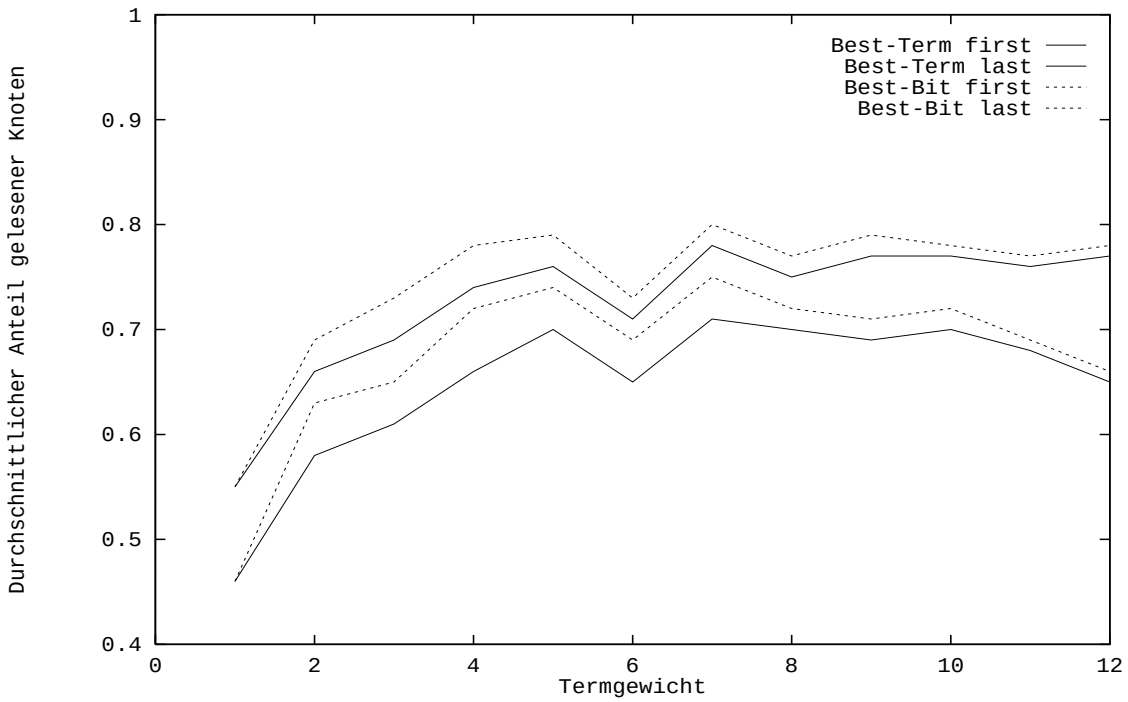


Abbildung 8.25: Besuchte Knoten bei $\varrho_{Best\text{-}Term}$ vs. $\varrho_{Best\text{-}Bit}$

unterstützt werden (was hier allerdings nicht betrachtet wird). Die nachfolgende Beschreibung orientiert sich im wesentlichen an der Darstellung in [Gonnet et al. 92].

8.4.5.0.4 Grundkonzepte

Im Gegensatz etwa zu anderen Zugriffspfaden (wie etwa invertierten Listen), die einzelne Dokumente streng voneinander trennen, wird beim Pat-Baum die gesamte Dokumentkollektion als ein einziger langer String aufgefaßt. Eine Folge strukturierter Dokumente hätte dann z.B. folgendes Aussehen:

```
Doc1() Doc2() Doc3( Ch1() Ch2()) Doc4(Tit() Abstr() Sec1( Subs1() Subs2())
Sec2())
```

(Die Berücksichtigung der Dokumentstruktur bei der Suche ist durch das Mitführen der entsprechenden Markup-Information als suchbares Element möglich.) Ein wichtiges Konzept für die Suche ist die semi-unendliche Zeichenkette (semi-infinite string, sistring). Ein **sistring** ist ein String, der an einer beliebigen Position im Text beginnt und sich im Prinzip unbegrenzt weit nach rechts ausdehnt, wie etwa im folgenden Beispiel:

```
01 - THIS IS A SAMPLE STRING
02 - HIS IS A SAMPLE STRING
03 - IS IS A SAMPLE STRING
04 - S IS A SAMPLE STRING
05 - IS A SAMPLE STRING
06 - IS A SAMPLE STRING
07 - S A SAMPLE STRING
07 - A SAMPLE STRING
...
```

Jedem sistring entspricht somit eindeutig eine Position im Text. Praktisch endet ein sistring spätestens am Textende, zudem kann man auch eine Maximallänge vorsehen. Für Textretrieval betrachtet man meist nur solche sistrings, die Wörter darstellenen, d.h. nur Positionen, denen ein Blank vorausgeht, und der sistring endet mit dem nächsten Blank.

Man definiert sich nun in der üblichen Weise eine lexikographische Ordnung auf den sistrings (also gilt etwa in unserem Beispiel ``A SA...'' < ``AMP...'' < ``E ST..'').

Ein **PAT-Baum** ist nun ein Patricia-Baum aller sistrings eines Textes. Dabei ist ein Patricia-Baum ein binärer Digital-Baum, bei dem die einzelnen Bits der Schlüsselwerte über die Verzweigung im Baum entscheiden (bei 0 nach links, bei 1 nach rechts). Zusätzlich wird bei einem Patricia-Baum noch bei jedem internen Knoten angegeben, welches Bit zum Verzweigen benutzt wird (entweder absolutüber die Bitposition oder relativ durch die Angabe der Anzahl übersprungener Bits). Dadurch können interne Knoten mit nur einem Nachfolger weggelassen werden. Die Schlüsselwerte werden als Textpositionen in den externen Knoten des Patricia-Baums gespeichert. Die inter-

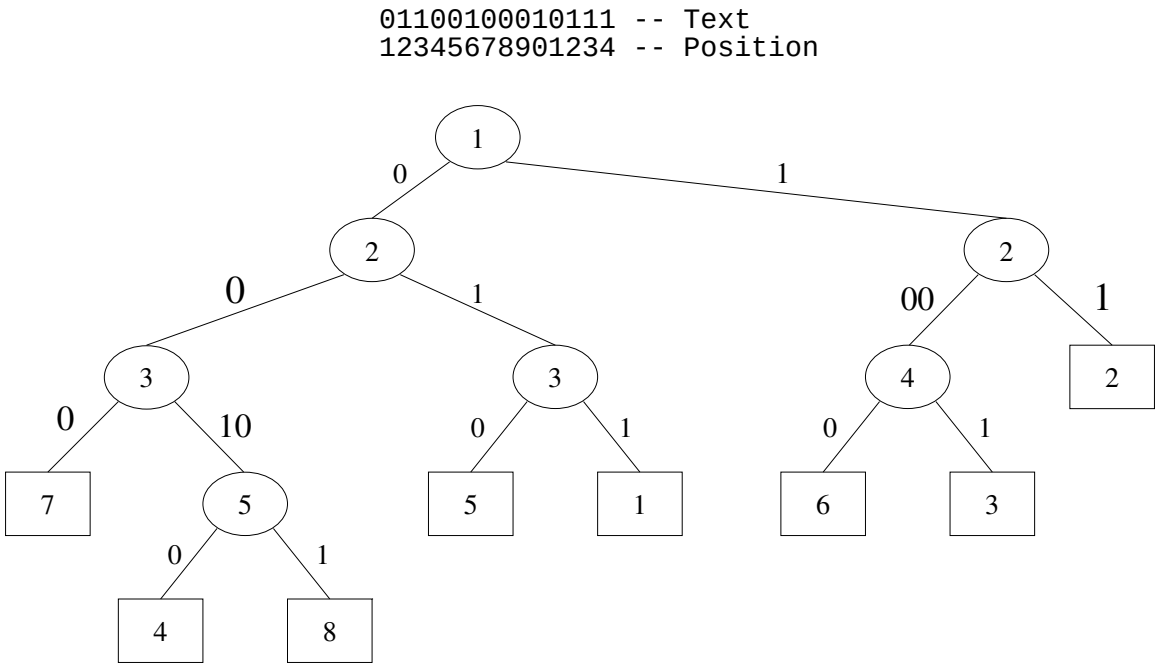


Abbildung 8.26: PAT-Baum nach dem Einfügen der ersten acht sistrings

nen Knoten enthalten nur die Angabe über das Verzweigungsbit sowie Zeiger auf die beiden Nachfolger. Ein Beispiel für einen PAT-Baum zeigt Abbildung 8.26.

8.4.5.0.5 Algorithmen auf PAT-Bäumen

PAT-Bäume erlauben vielfältige Arten der Suche, die im folgenden kurz skizziert werden. Dabei steht n für die Länge des zu indexierenden Textes

Präfix-Suche

Aufgrund der Definition des PAT-Baums enthält jeder Teilbaum alle sistrings die den gleichen Präfix besitzen. Somit muß man bei der Präfix-Suche von der Wurzel ausgehend einen Pfad entsprechend den Bits des Präfix' verfolgen, bis alle Bits des Präfix' aufgebraucht sind. Falls dabei Bits übersprungen wurden, muß man noch einen beliebigen sistring im Teilbaum auf den Präfix testen. Im positiven Fall bilden alle sistrings im Teilbaum zusammen die Antwortmenge, im negativen Fall ist die Antwortmenge leer.
Ein Beispiel für eine Präfix-Suche zeigt Abbildung 8.27. Hier liefert die Suche nach dem Päfix 100* den Teilbaum mit den externen Knoten 3 und 6.
Der Zeitaufwand für die Präfix-Suche ist offensichtlich höchstens $O(\log n)$, in der Regel wächst er linear mit der Länge des Präfixes.

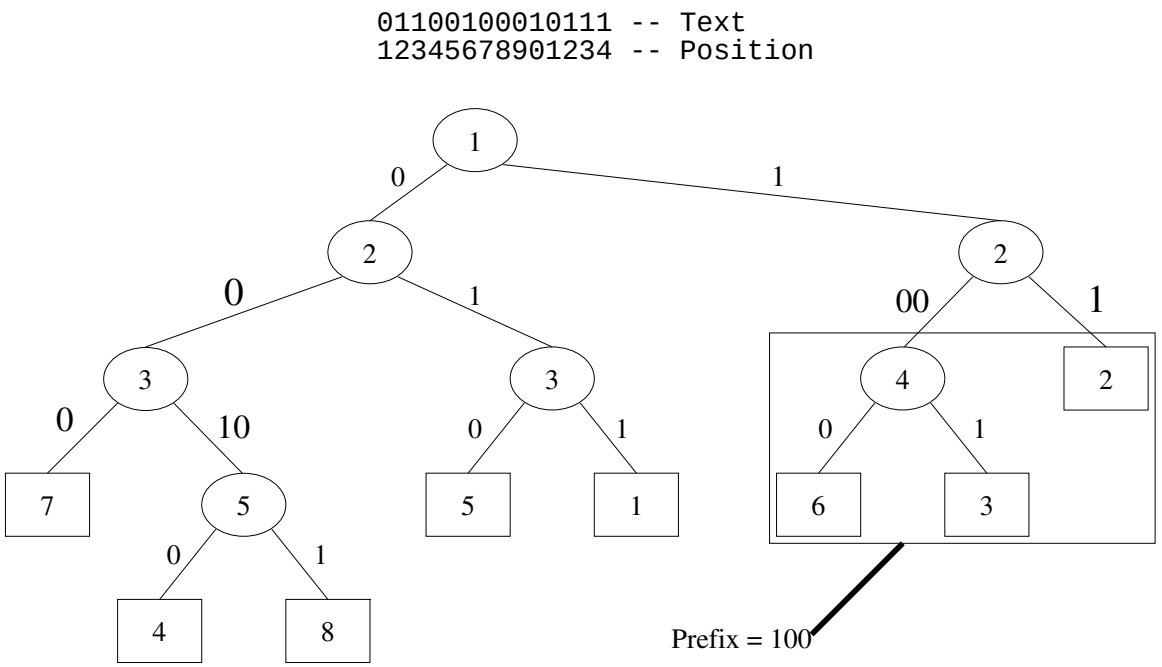


Abbildung 8.27: Präfix-Suche

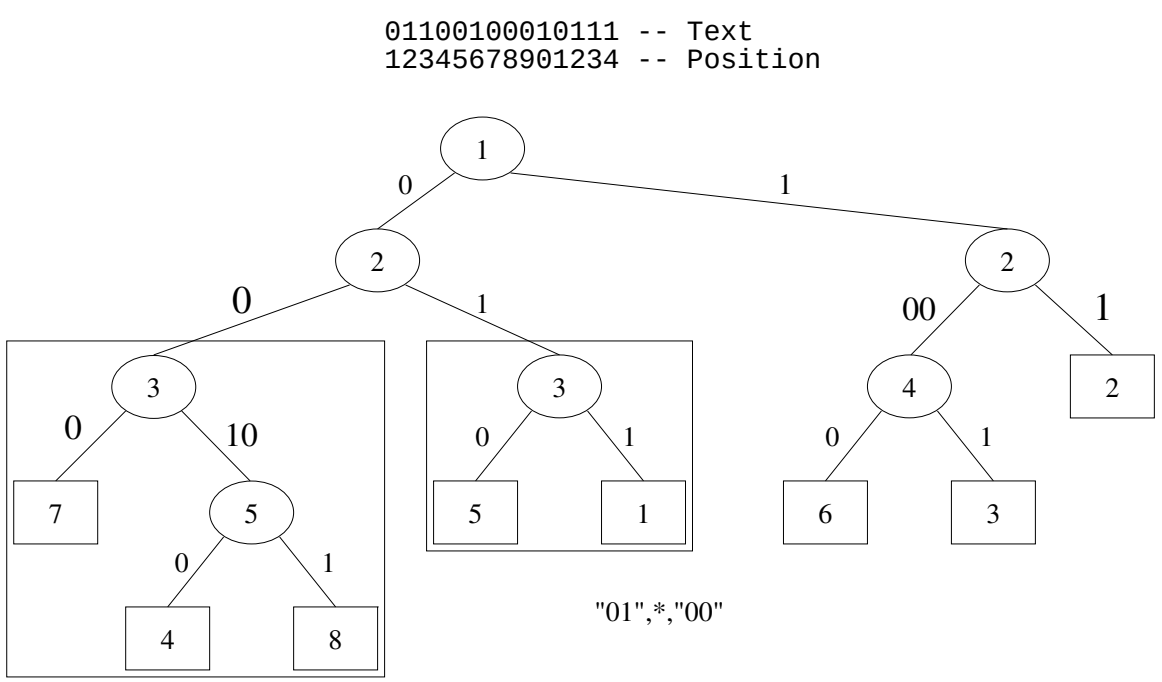


Abbildung 8.28: Reihenfolge-Suche

Reihenfolge-/Abstandssuche
Sucht man Wörter (oder Strings) s_1 und s_2 in einer bestimmten Reihenfolge (evtl. zusätzlich in einem bestimmten Maximalabstand), so muß man zunächst die beiden Wörter einzeln im PAT-Baum suchen und sortiert dann die kleinere der beiden Antwortmengen nach aufsteigender Position. Nun vergleicht man die Elemente der zweiten Antwortmenge mit jedem Element der ersten bezüglich des Einhaltens der Reihenfolge-/Abstandsbedingung. Sei m_1 die Größe der kleineren Antwortmenge und m_2 die der anderen, so ergibt sich ein Zeitaufwand von $O((m_1 + m_2) \log m_1)$

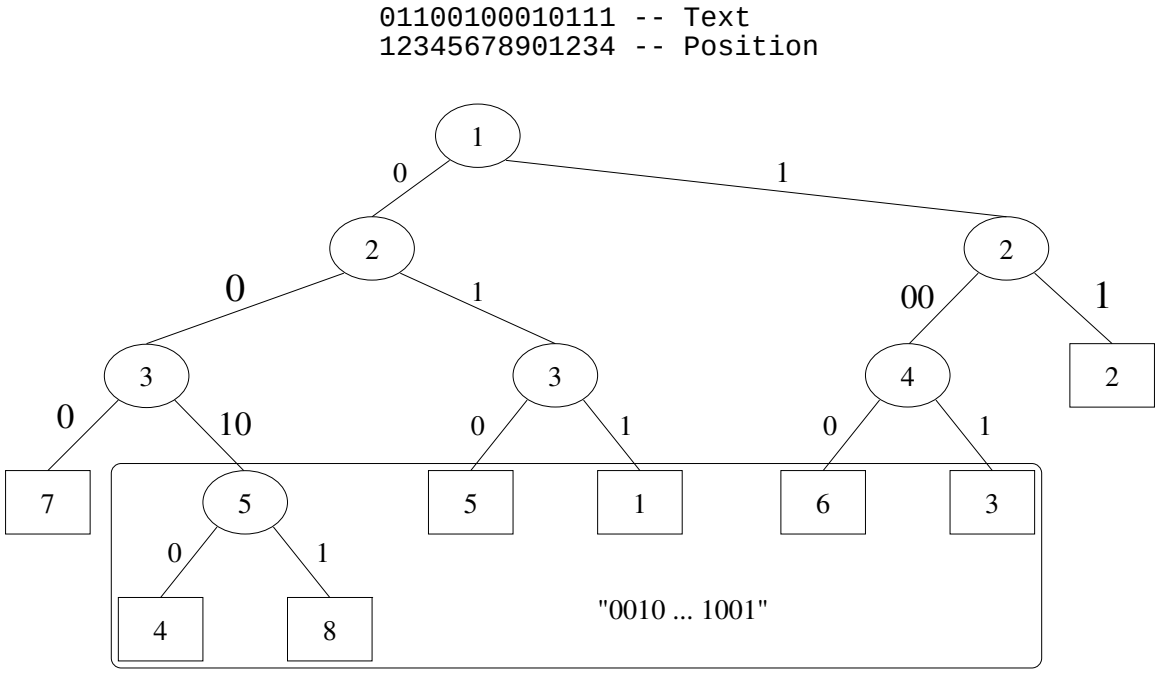


Abbildung 8.29: Bereichssuche

Bereichssuche
Bei der Bereichssuche werden die Vorkommen aller Wörter gesucht, die entsprechend der lexikographischen Ordnung zwischen zwei Wörtern liegen. Hierzu wird jedes der beiden Intervallgrenzwörter einzeln im PAT-Baum gesucht. Dann bilden die beiden zugehörigen Teilbäume sowie alle Teilbäume dazwischen die Antwortmenge. Ein Beispiel hierzu zeigt Abbildung 8.29. Der Zeitaufwand für diesen Algorithus beträgt offensichtlich $O(\log n)$.

Längste Wiederholung
Sucht man nach dem längsten sistring, der zweimal im Text vorkomt, so muß man nur den höchsten internen Knoten im PAT-Baum bestimmen (wobei zusätzlich evtl.

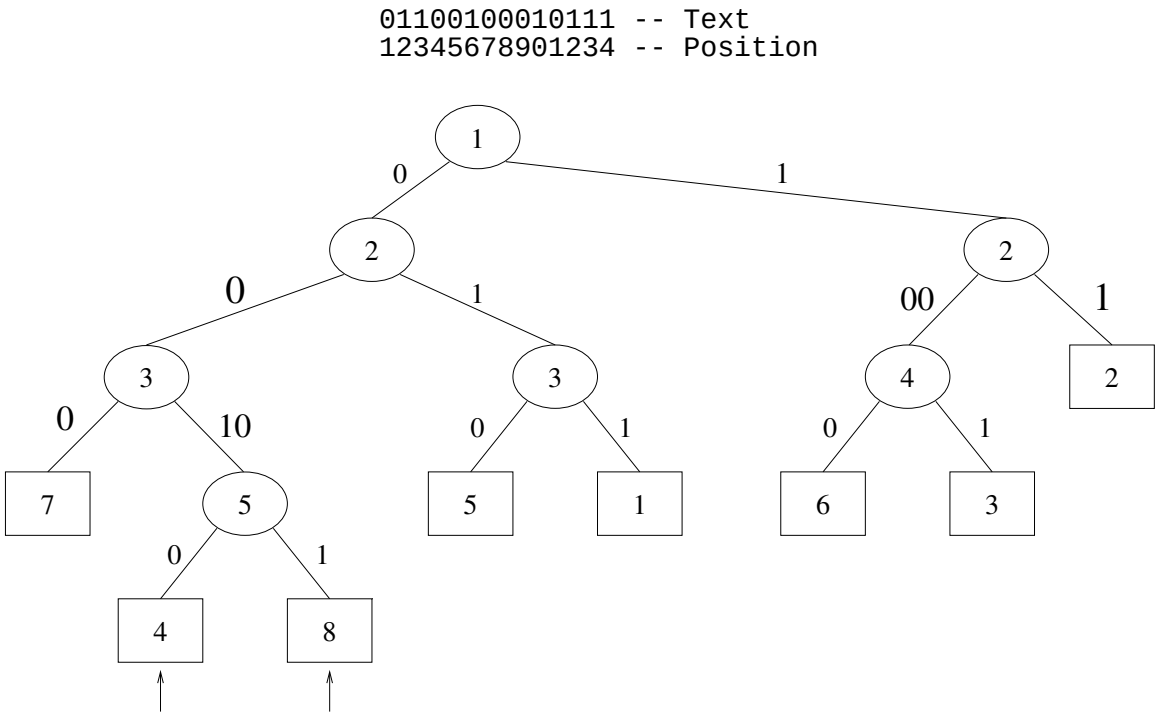


Abbildung 8.30: Suche nach der längsten Wiederholung

übersprungene Bits zu berücksichtigen sind). Diese Aufgabe kann bereits beim Aufbau des PAT-Baums gelöst werden. Ein Beispiel hierzu zeigt Abbildung 8.30.
Ein verwandtes Problem ist die Suche nach der längsten Wiederholung mit einem vorgegebenen Präfix, also in einem Teilbaum. Um den Teilbaum nicht vollständig absuchen zu müsen, kann bereits beim Aufbau des PAT-Baums bei jedem internen Knoten mit einem zusätzlichen Bit vermerken, auf welcher Seite sich der höhere Teilbaum befindet. Dann ergibt sich ein Zeitaufwand von $O(\log n)$.

Häufigkeitssuche
Sucht man nach dem häufigsten String mit einer bestimmten Länge (n-gram), der also am häufigsten im Text auftritt, so muß man nur denjenigen internen Knoten im entsprechenden Abstand zur Wurzel suchen, der in seinem Teilbaum die meisten Knoten enthält. (z.B. illustriert Abbildung 8.31 die Suche nach dem häufigsten Bigram, was als Antwort 00 liefert, das dreimal vorkommt). Hierzu muß der PAT-Baum traversiert werden, was einen Zeitaufwand von $O(n/a)$ erfordert, wobei a die durchschnittliche Antwortgröße ist. Alternativ dazu könnte man schon beim Aufbau des Baumes in jedem Knoten die Anzahl der Knoten im zugehörigen Teilbaum festhalten.
Sucht man nach dem häufigsten Wort im Text, so werden nur sistrings betrachtet, die mit Blank beginnen und enden; somit beschränkt sich die Suche auf den Teilbaum

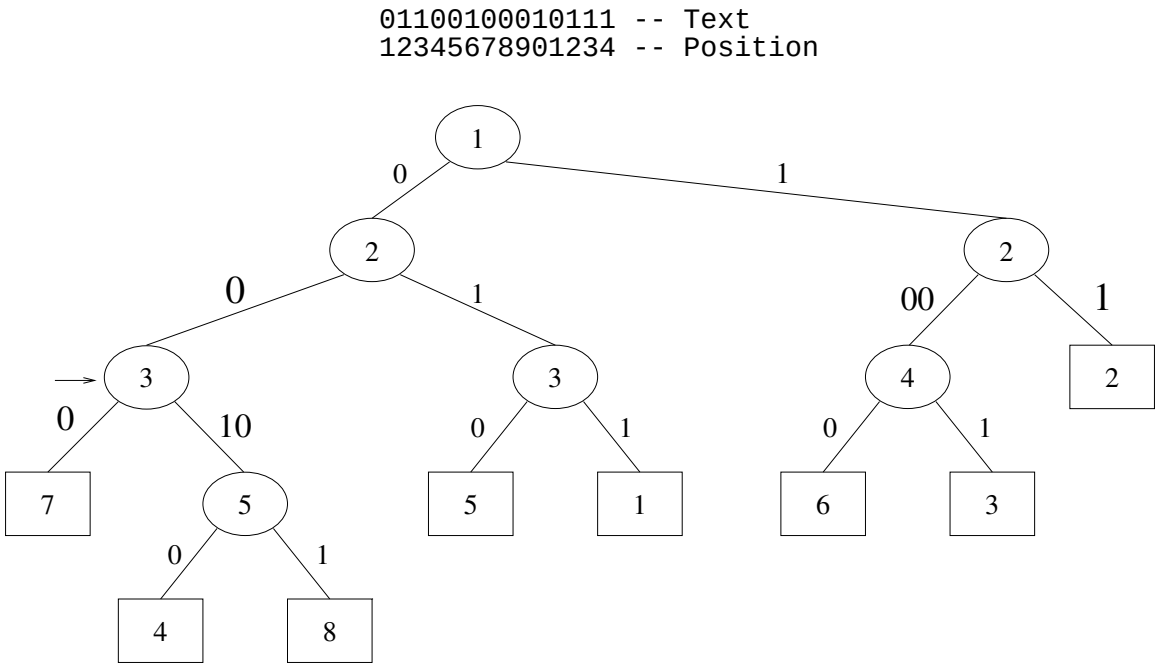


Abbildung 8.31: Suche nach dem häufigsten Bigram

der mit Blank beginnenden sistrings, und zudem wird beim Durchlaufen immer nur bis zum zweiten Blank gesucht.

Reguläre Ausdrücke

Zur Suche nach regulären Ausdrücken wird der Suchausdruck in einen endlichen Automaten übersetzt, und dieser wird dann auf dem PAT-Baum abgearbeitet. Abbildung 8.32 zeigt ein Beispiel hierzu. (Zu den Details siehe [Gonnet et al. 92]).

8.4.5.0.6 PAT-Arrays

Da für große Textmengen der PAT-Baum nicht komplett in den Hauptspeicher paßt, muß man nach effizienten Möglichkeiten suchen, um diese Datenstruktur auf dem externen Speicher zu halten. Hierzu wurden PAT-Arrays entwickeln, die zudem weniger Speicherplatz als PAT-Bäume benötigen — allerdings auf Kosten größerer Laufzeit für bestimmte Suchprobleme.

Der PAT-Array basiert auf der lexikographischen Reihenfolge der sistrings und enthält nur die zugehörigen Textpositionen (gemäß dieser Reihenfolge). Den PAT-Array zum PAT-Baum aus den vorangegangenen Beispielen zeigt Abbildung 8.33. Präfix- und Bereichssuchen können im PAT-Array als indirekte binäre Suche realisiert werden (da die Schlüsselwerte nicht im Array stehen, ist jeweils ein Zugriff auf den Text an der angegebenen Position notwendig). Diese Operationen erfordern dann $2 \log_2 n -$

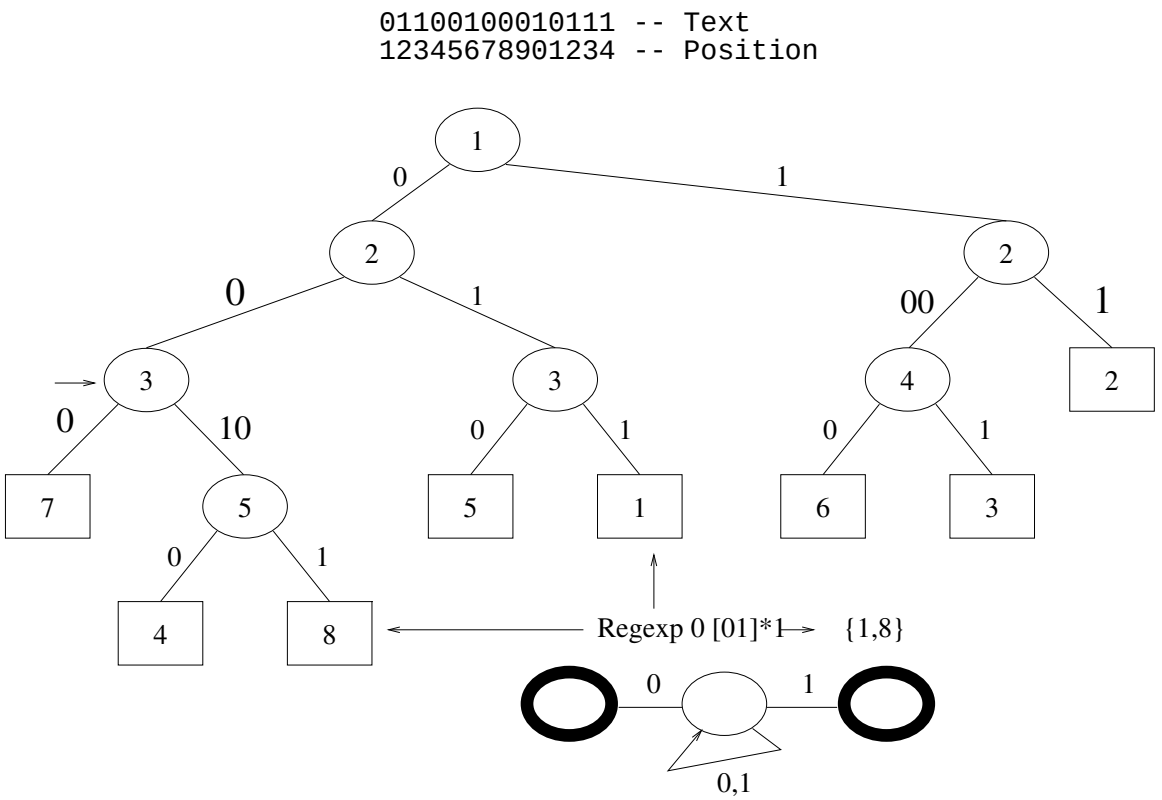


Abbildung 8.32: Suche nach regulären Ausdrücken

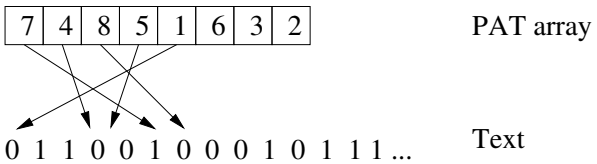


Abbildung 8.33: Beispiel eines PAT-Arrays

1 Vergleiche und $4 \log_2 n$ Plattenzugriffe. Andere Suchoperationen werden hingegen aufwendiger.

Chapter 9

IR and Databases

9.1 Introduction

The fields of databases (DB) and IR have evolved separately over a long period of time. This is mainly due to the difference in the major topics that have been studied: whereas IR has concentrated on dealing with natural language text (i.e. "unformatted data") and retrieval under uncertainty, the focus in databases is on data models, query languages, database consistency and efficiency. Uncertainty in database systems has been only a minor research topic (see e.g. [Lee 92], [Barbara et al. 92], [Prade & Testemale 84]).
In the following, we will show how the aspects of data structure and consistency can be applied to IR systems. It will become clear that future IR systems will have to integrate these concepts.
Figure 9.1 shows an example document from a typical IR database. It is obvious that besides (natural language) text, this document also contains fact data of different sorts (e.g. dates, names of persons and institutions). So users may want to ask for this information, too. That is, besides documents, they want to search for other types of objects, too. There may be two different goals behind such a need:

- In order to enhance document retrieval, users may want to extract information about different kinds of objects that occur in the current answer set, e.g. *Which index terms occur most frequently in the answer set?* or *Which authors occur in the answer set?*. Some current (commercial) IRS provide special commands for this type of queries.
- For some applications, there is a need to extract other information from the database, that is, the final answer consists of objects which are not documents. For example, a researcher may search for colleagues working in a certain field by asking *Which institutions / authors work in this field?*, or he might be interested in historical aspects by posing the query *When was most of the research in this area done?* Current IRS do not support this type of queries.

L2 ANSWER 1 OF 111
AN 87(01):411 NTIS Order Number: AD-A172 502/7/XAD
TI Controlling Inference. (Doctoral thesis)
AU Smith, David E.
CS Stanford Univ., CA. Dept. of Computer Science
NC Contract : N00014-81-K-0004
NR AD-A172 502/7/XAD; STAN-CS-86-1107
197 p. NTIS Prices : MF A01
Availability : Microfiche copies only.
PD 860400
LA English CY United States
OS GRA&I8701
AB Effective control of inference is a fundamental problem in Artificial Intelligence. Unguided inference leads to a combinatoral explosion of facts or subgoals for even simple domains. To overcome this problem, ...
CC 95F Bionics and artificial intelligence
CT *Artificial intelligence; Data bases; Decision making; Global; Information exchange; Problem solving; Efficiency *Inference; Control; Theses; Expert systems
UT NTISDODXA

Figure 9.1: Example document in a bibliographic database

So we see that there seems to be a need for providing querying facilities that enable an user of an IRS to ask not only for documents, but for all types of objects occurring in an IR database. This goal can be accomplished only by introducing a powerful data model and query language which supports this type of queries.

9.2 Basic DBMS features

In this section, the basic concepts of database management systems (DBMS) (see e.g. [Date 90], [Ullman 88]) are described, and it is shown how these concepts apply to IRS. Besides the data model, the different types of DB consistency are described:

- **Integrity** means that the static relationships between different items stored in a database are maintained (e.g. that for each account in a bank database, there must be an address record for the person owning the account).
- **Recovery** is the ability of the DBMS to recover after a system failure and to return to a consistent database state.
- **Concurrency control** guarantees DB consistency when multiple users access the database, even when they perform concurrent updates.
- **Security** is a feature to restrict the rights of certain users to perform certain operations on specific parts of the database, e.g. in order to make portions of the database inaccessible for some users.

9.2.1 Data model

In general, a data model fulfills two tasks:

1. It describes the structure of the data along with inherent relationships between different portions of data. For a specific database, the corresponding schema based on the data model ensures that only data confirming to the schema is stored in the database.
2. Along with the data model comes a powerful query language which allows to ask for any item stored in the DB.

Three different data models will be described in sections 9.3 and 9.4

9.2.2 Integrity

Database integrity means that the inherent relationships between different items stored in the DB are maintained.
As an example, regard the database shown in figure 9.2. The database consists of two tables, a document and a dictionary table. The document table contains for each document its number and the set of terms occurring in the document text. In the dictionary table, we have all the terms from the documents of the DB along with

DOCS	DOCNO	TERMS
	d_1	t_1, t_2, t_3
	d_2	t_2, t_3

DICT	TERM	DOCFREQ
	t_1	1
	t_2	2
	t_3	2

Figure 9.2: Example IR database with document and dictionary table

DOCS	DOCNO	TERMS
	d_1	t_1, t_2, t_3
	d_2	t_2, t_3
	d_3	t_1, t_3, t_4

DICT	TERM	DOCFREQ
	t_1	2
	t_2	2
	t_3	3
	t_4	1

Figure 9.3: DB after insertion of d_3 with terms $\{t_1, t_3, t_4\}$

their number of occurrences. So we have the integrity rule that the content of the DOCFREQ column should always contain the actual number of documents in the DB in which a term occurs. If we now add a document d_3 with terms $\{t_1, t_3, t_4\}$, then the database should have the content shown in figure 9.3. A DBMS monitors the integrity rules specified for a certain DB, and it forbids changes that invalidate integrity. However, it should be noted that current commercial DBMS only provide minimum support for the inforcement of integrity rules.
Now let us look at the following example application:

```
procedure newdoc(docno,terms);
insert(DOCS,<docno,terms>);
foreach t in terms do
  if exists(DICT,<t,?f>)
    then update(DICT,<t,f+1>)
    else insert(DICT,<t,1>);
```

The `procedure newdoc(docno,terms)` adds a new document with the number `docno` and the set of terms `terms` to the DB. First, the operation `insert(DOCS,<docno,terms>` adds the corresponding row to the DOCS table. Then each term `t` is looked up in the dictionary table via the operation `exists(DICT,<t,?f>)`, returning a boolean value of FALSE if there is no corresponding entry, and TRUE along with the current term frequency in `f`. Then we either update the current entry, or we insert a new row with frequency 1. Now let us apply this procedure for the insertion of the document d_3 from above. After the `insert` operation, we have the DB state shown in figure 9.4. We see that this state is inconsistent, since the DICT table does not yet reflect the change in the DOCS table. It is obvious that it is not possible to add new documents to the database without violating the integrity rules temporarily.
This observation leads to the introduction of the concept of a **transaction**: A trans-

DOCS	DOCNO	TERMS
	d_1	t_1, t_2, t_3
	d_2	t_2, t_3
	d_3	t_1, t_3, t_4

DICT	TERM	DOCFREQ
	t_1	1
	t_2	2
	t_3	2

Figure 9.4: DB state after `insert(`d_3`,`$\{t_1, t_3, t_4\}$`)`

action is a change from one consistent state of the DB to another. An application has to specify the beginning and the end of a transaction in the form `begin transaction` — database operations — `end transaction`. The DBMS checks the integrity rules after the end of transaction. If a violation is detected, then all changes performed by the transaction are undone.

9.2.3 Recovery

Via the concept of recovery, a DBMS guarantees a stable storage of consistent DB states, even after system failure. With respect to transactions, this means that the effect of all incomplete transactions is undone, and the changes performed by all completed transactions are stable. Most DBMS use a log file for writing logging information about transactions to disk. When restarting after a system failure, the DBMS uses this logging information in order to perform the necessary changes in the DB.

9.2.4 Concurrency

In the past, typical IR applications were such that only the administrator of the database was allowed to perform database updates, and all other users did only retrieval. Today, e.g. in office information systems, several users are allowed to change the content of the IR DB by adding or deleting documents or by updating the content of a document. This type of applications raises the need to run transactions should be allowed to run in parallel, in order to increase the throughput of the system. However, the parallel execution of transactions poses new problems, since inconsistent DB states may occur.

Figure 9.5 shows an example for the parallel execution of two transactions. After transaction T_1 got the information that term t has a frequency of 1, transaction T_2 looks up the same entry, before T_1 can update the frequency information. This leads to the effect that the frequency of t is increased incorrectly only by 1, whereas a sequential processing of the two transactions would lead to an increase of 2. In order to avoid this type of effect, a DBMS needs facilities for **concurrency control**. This component synchronizes concurrent transactions such that effect is the same as that

	transaction
T_1	T_2
...	...
exists(DICT,<t,1>	
	exists(DICT,<t,1>
update(DICT,<t,2>	
	update(DICT,<t,2>

Figure 9.5: Parallel execution of two transactions

DOCS	DOCNO	SCTY	TERMS
	d_1	1	t_1, t_2, t_3
	d_2	1	t_2, t_3
	d_3	2	t_1, t_3, t_4

Figure 9.6: Documents with security levels

of single-user operation. As a means for performing concurrency control, most DBMS use locks on items of data in order to prohibit concurrent access to the same items.

9.2.5 Security

In typical DB applications, there are different groups of users, where each group may perform only specific operations on certain parts of the database. As a simple application of this concept to IR, assume that each document is assigned a security level, where user u_1 is allowed to read only documents with SCTY=1 and user u_2 may read all documents.
Figure 9.6 shows the documents currently stored in the DB, which all can be seen by user u_2 as shown in figure 9.7. User u_1 is not allowed to see document d_3, and so he also should only see a view of the dictionary which is consistent with the documents he is allowed to look at (figure 9.8). Otherwise, u_1 would be able to draw conclusions from the dictionary about the content of the documents which he is not allowed to see.
Today's DBMS mostly can restrict the views of users only to certain rows or columns

DOCS	DOCNO	TERMS
	d_1	t_1, t_2, t_3
	d_2	t_2, t_3
	d_3	t_1, t_3, t_4

DICT	TERM	DOCFREQ
	t_1	2
	t_2	2
	t_3	3
	t_4	1

Figure 9.7: View of user u_2

DOCS	DOCNO	TERMS
	d_1	t_1, t_2, t_3
	d_2	t_2, t_3

DICT	TERM	DOCFREQ
	t_1	1
	t_2	2
	t_3	2

Figure 9.8: View of user u_1

BOOKS	AUTHORS	TITLE	PRICE	TERMS
	A_1, A_2	T_A	10	T_1, T_2
	A_2	T_B	20	T_1, T_2
	A_1	T_C	10	T_1, T_2, T_3

Figure 9.9: A relation not in 1st ormal-form

of tables, so it would not be possible to achieve the type of behaviour from our example with this type of systems. A possible solution would be to store the dictionary table not explicitly, but to derive the dictionary data the user is asking for dynamically (by considering the security information).

9.3 Modeling structure

In this section, we discuss two different data models with respect to their application to IR problems. First, we present the most popular model, namely the relational model, and point out some weaknesses of this model. The second model, the NF^2 model overcomes most of these shortcomings.

9.3.1 Relational model

The relational model is based on the concept of a table, which is called a relation here. A single column of a relation is called an attribute. Mathematically, a relation is a set of tuples. (see e.g. [Macleod 91], [Blair 88], [Lynch 91] for the combination of the relational model with IR systems).

9.3.1.1 Normalization

As a basic requirement, a relation must be in **1st normal form**. This means that all attribute values must be atomic. As an example, consider the table shown in figure 9.9. It contains the set-valued attributes AUTHORS and TERMS, and so it is not in 1st normal form.

BOOK	BNO	TITLE	PRICE
	1	T_A	10
	2	T_B	20
	3	T_C	10

AUTHORS	BNO	AUTHOR
	1	A_1
	1	A_2
	2	A_2
	3	A_1

TERMS	BNO	TERM
	1	T_1
	1	T_2
	2	T_1
	2	T_2
	3	T_1
	3	T_2
	3	T_3

Figure 9.10: Result of the normalization step

In order to achieve 1st normal form, we have to perform a normalization, which distriutes the data from the BOOKS table across the three relations BOOK, AUTHORS and TERMS shown in figure 9.10. For each set-valued attribute from the BOOKS table, we have to form a separate relation.

9.3.1.2 SQL

The standard query language for relational databases is SQL. It is a descriptive language, i.e. a SQL statement specifies only what the result should be like, but not how it is to be computed. This takes some burden from the user, and it also offers the possibility for the DBMS to choose an efficient way for producing the result.
The general form of an SQL statements is as follows:
`select` < attributes>
`from` < relations>
`where`<expression>
In the `select` clause, we specify the attributes that make up the output relation. Following the `from` clause, we have to name the relations to which we refer to. In the `where` clause, conditions are specified that have to be fulfilled by the tuples in order to go into the result.
In the following examples, we refer to the relations shown in figure 9.10. The statement

```
select * from BOOK
```

yields the complete BOOK relation (here * stands for all attributes of the named relation). If we want to select only certain tuples, we have to specify a condition in

the `where` clause, e.g.

```
select *
from BOOK B
where B.PRICE > 10
```

If we want to see only certain attributes of the selected tuples (a so-called projection), we have to name each attribute to be displayed in the `select` clause, e.g.

```
select B.TITLE from BOOK B
```

For clarity, we use so-called tuple variables here (like B in this example) which are declared in the `from` clause by following the relation name they refer to. Then we can use them in the `select` and the `where` clause in order to make it clear to which relation we refer to. A reference to an attribute then takes the form <attribute>.<variable>. Since the normalization process has distributed our data across several relations, we need an operation that combines the data from two relations. This so-called join operation is achieved by simply naming more than one relation in the `from` clause. For example, the following statement retrieves the titles of books indexed with term T1:

```
select B.TITLE
from BOOK B, TERM D
where D.TERM = 'T1'
and B.BNO = D.BNO
```

9.3.1.3 Boolean retrieval with SQL

We now show how Boolean retrieval (i.e. selection of documents containing certain terms as specified by a Boolean expression) can be performed with SQL. A search for titles of books indexed with terms T1 and T2 is formulated as follows:

```
select B.TITLE
from BOOK B, TERMS X, TERMS Y
where X.TERM = 'T1' and B.BNO = X.BNO and Y.TERM = 'T2' and B.BNO = Y.BNO
```

Here we need two tuple variables for the TERMS relation, since we search for one tuple containing term T1 and a different tuple with T2; in addition, both tuples must contain the same BNO, and then we retrieve the corresponding title with this BNO from relation BOOK. To go one step further, let us search for titles of books indexed with T1 and (T2 or T3):

```
select B.TITLE
from BOOK B, TERMS X, TERMS Y, TERMS Z
where X.TERM = 'T1' and B.BNO = X.BNO
and (Y.TERM = 'T2'  and B.BNO = Y.BNO or Z.TERM = 'T3'  and B.BNO = Z.BNO)
```

We see that already these simple Boolean queries lead to rather complex SQL statements.

9.3.1.4 Querying for other types of objects

On the other hand, SQL allows us to ask for any type of object in the DB, as will be shown in the following. For example, a user might want to know the names of authors who write about T1, i.e search for authors whose books are indexed by TERM T1:

```
select A.AUTHOR
from AUTHORS A, TERMS D
where D.TERM = 'T1' and A.BNO = D.BNO
```

As a second example, consider the query *What topics are covered by cheap books?*, i.e. *List terms assigned to books with PRICE < 20*:

```
select D.TERM
from TERMS D, BOOK B
where B.PRICE < 20 and D.BNO = B.BNO
```

9.3.1.5 Advantages and disadvantages of the relational model

+ The relational model is based on a small number of concepts and commands.
+ The query language SQL is more powerful than most IR query languages, since it allows to ask for any type of object in the DB. Furthermore, the result of a query is always a relation. This allows for a nesting of queries (similar to Boolean query languages of commercial IRS where one query can refer to the result of another).
– Even simple IR queries lead to complex SQL statements. For this reason, SQL would not be feasible for a user interface to an IRS based on the relational model.
– The requirement that relations must be in 1st normal form has the effect that IR objects are spread across several relations. Besides making query formulations more complex, the output of a query in the form of a relation is not appropriate. Finally, joining the data of an object together is computationally rather expensive, so normalization leads to a loss in efficiency.

9.3.2 NF^2 model

Some of the disadvantages of the relational model are overcome by the non-first-normal-form (NF^2) model (see [Schek & Pistor 82] and [Desai et al. 87] for the application of this model to IR). This model does not require that relations are in first normal form. Instead, attributes may be relations (sets of tuples) again. So we have nested relations in general. This allows for an adequate representation of IR objects, where a document can be stored as a single tuple of a (nested) relation.

BOOKS	AUTHORS	TITLE	PRICE	TERMS
	A_1, A_2	T_1	10	T_1, T_2
	A_2	T_1	20	T_1, T_2
	A_1	T_1	10	T_1, T_2, T_3

Figure 9.11: A NF^2 relation

9.3.2.1 NF^2 relations and query language

A simple example of a NF^2 relation is the BOOKS relation shown in figure 9.11 The query language SQL-NF^2 for the NF^2 model is similar to SQL, with the following extensions: We may specify set constants as operands, and Boolean expressions may be formed with operators for set comparison and element inclusion. As a simple example, the following query retrieves the titles of books indexed with term T1:

```
select TITLE from BOOKS
where TERMS ∋ 'T1
```

9.3.2.2 Boolean retrieval with SQL-NF^2

We now show how Boolean retrieval can be formulated in SQL-NF^2. A query searching for the titles of books indexed with terms T1 and T2 is formulated as

```
select TITLE from BOOKS
where TERMS ⊇ { 'T1','T2'} .
```

Searching for titles of books indexed with T1 and (T2 or T3) is accomplished via the statement

```
select TITLE from BOOKS
where TERMS ∋ 'T1' and (TERMS ∋ 'T2' or TERMS ∋ 'T3')
```

or alternatively

```
select TITLE from BOOKS
where TERMS ⊇ { 'T1','T2'} or TERMS ⊇ { 'T1','T3'}
```

As an even more complex example, consider a search for the title and price of books indexed with T1 and T2 and written by author A2:

```
select TITLE, PRICE from BOOKS
where TERMS ⊇ { 'T1','T2'} and AUTHORS ∋ 'A2'
```

9.3.2.3 Retrieval with weighted document indexing

So far, we have considered only a fairly simple NF^2 relation where the attributes are sets of atomic values. As a real example of nested relations, consider the BOOKSW relation shown in figure 9.12. This relation represents documents with weighted index terms. The attribute WTERM of the BOOKSW relation is a relation again, with

BOOKSW

BNO	AUTHORS	TITLE	PRICE	WTERM	
				WGT	TERM
1	A_1, A_2	T_1	10	W_1	T_1
				W_2	T_2
2	A_2	T_1	20	W_3	T_1
				W_1	T_2
3	A_1	T_2	10	W_2	T_1
				W_1	T_2
				W_3	T_3

Figure 9.12: A NF^2 relation representing weighted document indexing

the attributes TERM containing the index term WGT containing the corresponding weight.
In retrieval, we can refer to the indexing weights by posing additional conditions for the corresponding attribute WGT. For example, the following statement searches for titles of books written by author A1 and with a weight for term T1 of at least 0.5:

```
select TITLE from BOOKSW
where AUTHORS ∋ 'A1' and
(select * from WTERM
where TERM = 'T1'
and WGT ≥ 0.5) ≠ {}
```

The query language also offers a number of additional operators that are useful for referring to the indexing weights:

CARD gives the cardinality of the relational argument (the number of tuples in it).
SUM takes as argument a set of numeric values and returns the sum.
AVG computes the arithmetic average,
MAX the arithmetic maximum and
MIN the arithmetic minimum of a set of numeric values.

With these operators, we can e.g. search for titles of books containing at least two of the terms T1, T2, T3 with a weight > 0.5:

```
select TITLE from BOOKSW
where CARD( {T1,T2,T3} ∩
(select TERM from WTERM
where WGT > 0.5)) ≥ 2
```

It is also possible to achieve a simple type of ranking by using the order clause, which sorts a set of tuples according to the values of a specified attribute. As an example, the following statement searches for titles of books containing the terms T1, T2, T3 and ranks the result tuples by the decreasing sum of the corresponding weights:

```
select TITLE from BOOKSW
```

```
order by decreasing SUM(
select WGT from WTERM
where TERM = 'T1' or TERM = 'T2' or TERM = 'T3')
```

An alternative query formulation uses wildcards for weights:

```
select TITLE from BOOKSW
order by SUM(
select WGT from({<T1,*>,<T2,*>,<T3,*>}∩ select * from WTERM))
```

9.3.2.4 Advantages and disadvantages of the NF² model

+ The NF² model fits to IR applications, since documents can be represented within a single (nested) relation.
+ The query language is powerful, and most IR queries require only simple query statements.
– It is possible to handle indexing weights in query statements, but this leads to complex query formulations. From the IR point of view, weights should be part of the model, so that there is no need to address them explicitly.

9.4 Object-oriented databases

The data models described in the previous section address only model the structure of the data in a DB. In addition to that, object-oriented database management systems (OODBMS) also allow for modelling the behaviour of the database objects (see e.g. [Bertino & Martino 91]. In the following, we first describe the structural aspects and then the behavioural aspects of this approach (see also [Harper & Walker 92] for the integration of OODBMS and IR concepts).

9.4.1 Modeling structure

9.4.1.1 Object classes

A typical problem in many IR systems is the varying structure of the documents to be stored in the DB, where different types of documents contain different attributes. As a simple example, assume that we have the document types article, book and inbook (for a single article in a book, e.g. in proceedings) with the following attributes:

- article: authors, title, year, journal title, volume, number, pages, terms
- book: authors, title, year, publisher, terms
- inbook: authors, title, year, booktitle, editors, pages, publisher, terms

There is no appropriate solution for modelling these three document types in the relational or the NF² model: Either we have to use three different relations for the different document types, or we can declare one relation with the attributes common

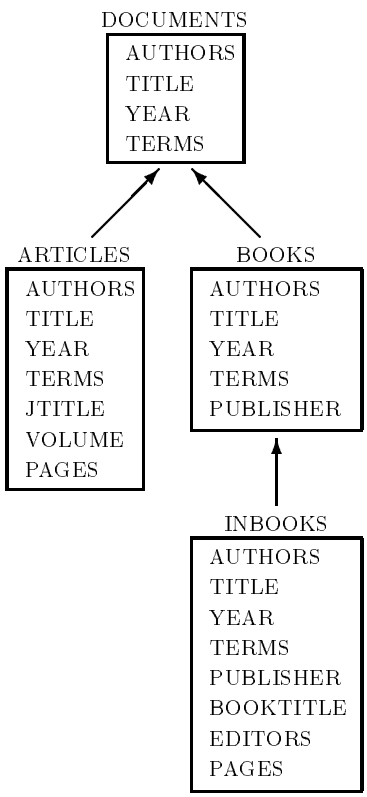


Figure 9.13: Class hierarchy for document types

to all types, and three more relations for the additional attributes. For solving this problem, OODBMS offer the concept of **object classes** in combination with an **inheritance** mechanism. An object class consists of a set of objects with identical structure. Between two object classes, an inheritance relationship may be declared. In this case, the subclass inherits all attributes of its superclass. As an example, consider the class hierarchy shown in figure 9.13. Here DOCUMENTS is the most general superclass containing attributes common to all types of documents. It has the direct subclasses ARTICLES and BOOKS. INBOOKS is a subclass of BOOKS, since it comprises all attributes of the class BOOKS plus three more attributes.
A major advantage of this class hierarchy shows up in retrieval, where a query may relate to any subtree of the class tree. The only restriction is that all attributes named in the query must occur in the root of the subtree.
The following statement lists the titles of documents published after 1980 and indexed with T1 and T2:

```
select TITLE from DOCUMENTS
where YEAR > 1980 and TERMS ⊇ { 'T1','T2'}
```

Since we refer to the root of the document class tree, the query retrieves all types of documents (articles as well as books and inbooks) fulfilling the criteria. As an example query referring to a subtree, consider the following query searching for the titles of books or inbooks published by Springer and indexed with T3:

```
select TITLE from BOOKS
where PUBLISHER = 'Springer' and TERMS ∋ 'T1'
```

9.4.1.2 Complex objects

For modelling the structure of data, OODBMS allow for the combination of nested objects (like in the NF^2 model) with class hierarchies. Figure 9.14 shows an example for this combination: a document contains a set of parts, which in turn may contain a set of figures. A figure may be either a graphics, an image or a table. A specific document of this type is shown in figure 9.15

9.4.1.3 Object versions

Another important concept of OODBMS are **object versions**. When documents are created with the help of the computer (e.g. by using word processors and drawing tools), it is typical that different versions of the document occur over time. In some cases, it may be useful to be able to access previous versions. The versioning mechanism of OODBMS supports different versions for a single object. As an example, consider the document d1 shown in figure 9.16. This document consists of two parts p1 and p2, where p1 contains the figures f1 and f2. By altering figure f1/V1, we get f1/V2. This change propagates to p1, thus yielding the version p1/V2, and furtheron

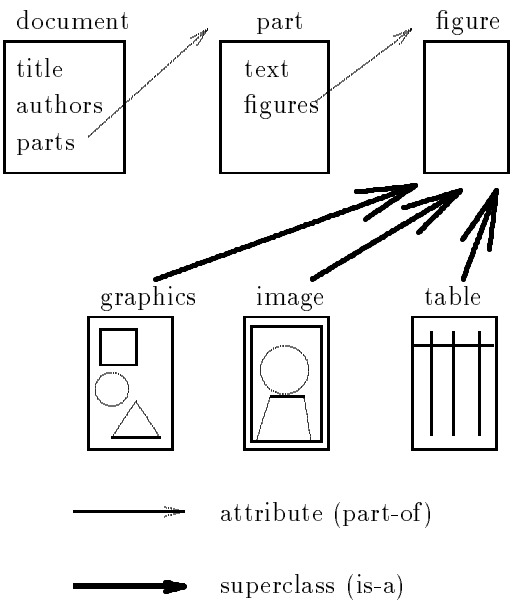


Figure 9.14: Documents as complex objects

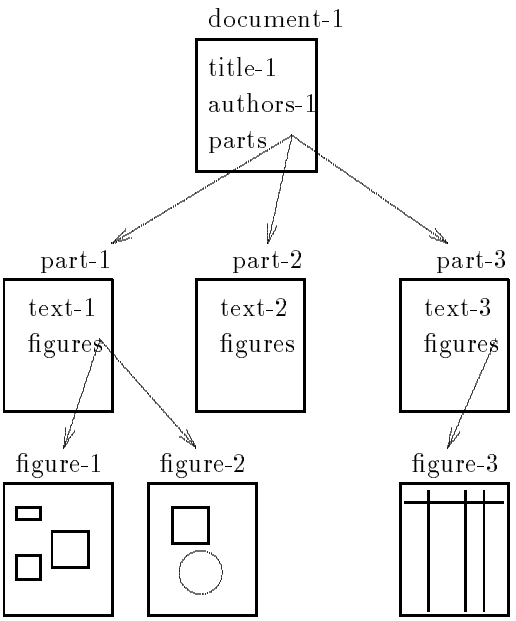


Figure 9.15: Structure of a specific document

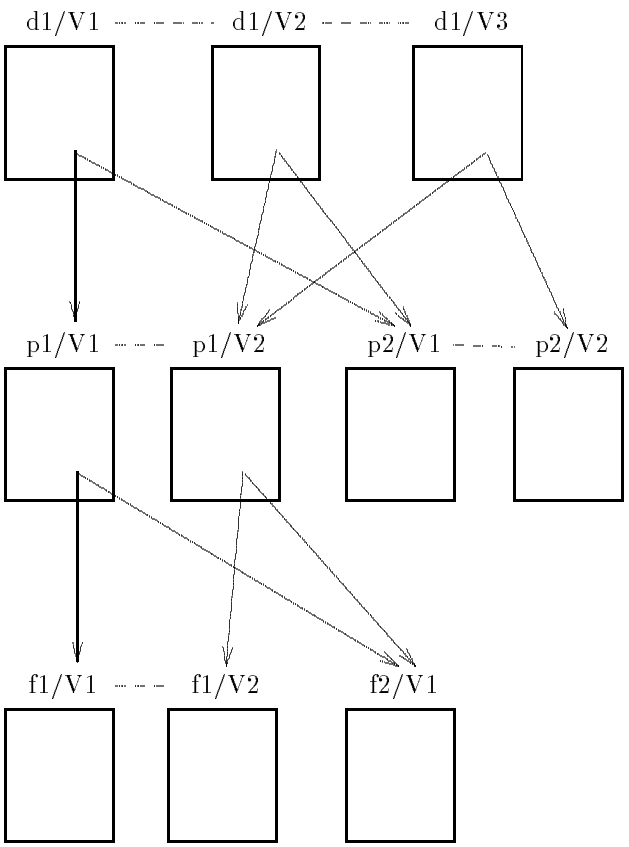


Figure 9.16: Object versions of a document and its subobjects

to d1, where we get the version d1/V2. Next, the text of p2 is changed, so we get p2/V2 and on the document level d1/V3.

9.4.2 Modeling behaviour

The object-oriented paradigm treats objects as a combination of object structure and behaviour. When a new object class is defined, in addition to the declaration of the structure of the objects also the operations that can be performed on these objects have to be defined. These operations are called methods. A method can be invoked only by sending the corresponding message to the object. This mechanism is based on the concept of **encapsulation**: The attribute values of an object cannot be accessed directly by other objects, only via sending appropriate messages. So the internal structure of an object is hidden, and reading attribute values as well as update or deletion of objects can be performed only via sending messages to the object.
As an example for the application of this concept in an IR system, assume that we have the object classes document and dictionary. The system performs automatic indexing of document texts, where the indexing procedure is a method of the class document which is invoked internally. The method

```
newdoc(text:string) -> document;
```

creates a new document with the given text, performs an automatic indexing and updates the dictionary. As a result, a new document object is returned. Sending the method

```
deletedoc();
```

to a document destroys the document object and performs the corresponding updates in the dictionary. In a similar way, calling the method

```
changedoc(text:string);
```

alters the text of the document and performs a re-indexing along with the necessary dictionary changes.
For accessing the attribute values of a document, we have the methods

```
doctext() -> string;
terms() -> set of string;
```

returning the text or the set of assigned terms, respectively. We may ask if a certain term is assigned to a document by sending the message

```
assigned(term: string) -> boolean;
```

with the term as argument.
The methods of the document class call the following methods of the dictionary class:

```
addposting(term: string; doc: document);
delposting(term: string; doc: document);
```

for adding or removing a posting to/from the dictionary. The dictionary object internally performs the necessary changes, i.e. creating/removing an entry for a certain term or increasing / decreasing its frequency. In addition, it also maintains a list of

the documents in which a term occurs. This information can be accessed for retrieval by means of the methods

```
nopostings(term: string) -> integer;
postings(term: string) -> set of document;
```

If the system provides set operations, boolean retrieval can be implemented easily by means of set operations on the results of the corresponding invocations of the `postings` method.

The concept of encapsulation offers the following advantages:

- There is no distinction between original attributes (e.g. document text) and derived attributes (e.g. assigned terms). For other objects, both types of attributes look the same.
- The implementation of methods (e.g. indexing algorithm) is hidden in the class definition and thus closely linked to the class. This is in contrast to other data models where algorithms have to be implemented in the application programs.
- Methods can be used for implicitely maintaining the consistency of the database (e.g. for updating the dictionary according to changes in the documents).
- Methods may be implemented differently for different classes (e.g. indexing of short articles vs. book indexing), but they look identical from outside. This feature is called **method overloading**

Chapter 10

Fact Retrieval

In this chapter, a new probabilistic approach to fact retrieval and its integration with text retrieval is described. Traditionally, a specific application of fact retrieval, namely statistical databases, always has been regarded a subfield of IR. We will not refer to this kind of application here. The interested reader is referred to [Staud 90], for example. Here we will discuss the problem of vague queries and imprecise data in fact databases, before we turn to the aspect of integrating text retrieval.

10.1 A Probabilistic Approach to Fact Retrieval

10.1.1 Introduction

In most of today's data base management systems (DBMSs), the query language is based on two-valued logic (e.g. relational algebra). This concept implies that for every object stored in a data base, a binary decision can be made by the system whether the object is an answer to the current request or not. Based on this feature, efficient query processing strategies are possible.
On the other hand, handling of user requests that cannot be expressed in two-valued logic is difficult with current DBMSs:

- In engineering applications, when a new part has to be developed, it is often more effective to start with a similar part already constructed, instead of developing the new part from scratch. If the parts are stored in a database, it should be possible to search for parts similar to a given specification. The system described in [Schneider et al. 89] for this purpose is based on a relational database for the retrieval of a set of candidate objects. Then for each object in this set its similarity to the specification is computed.
- In materials data systems [Westbrook & Rumble 83], a large number of requests either seek for materials similar to a known material or for materials that are optimum with respect to a number of criteria. As there is also a large number

of missing values for the materials attributes, systems based on Boolean query logic rarely can provide satisfactory answers [Ammersbach et al. 88].

- Business decision making very often means to find an optimum solution with respect to a number of criteria. If the decision relates to items stored in a database, then there should be a method that retrieves items close to the optimum (see the example below).

In all these applications, the query languages of current DBMSs offer little support. Mostly, users are forced to submit a series of queries in order to retrieve some objects that are possible solutions to their problem. Moreover, they often cannot be sure if they tried the query that retrieves the optimum solution.

In the field of information retrieval, similar problems have been investigated for a long time.[1] For this kind of queries, ranking methods have been developed which yield a ranked list instead of a fixed set of documents as an answer to a query. It has been shown in experiments that ranking methods yield significantly better results than search methods based on Boolean logic [Salton et al. 83] [Fuhr 86].

Text retrieval systems with ranking first seek for documents that contain terms from the query. A relevance status value (RSV) is computed for each document, and then documents are ranked according to descending RSVs. For this task, two major theoretical models have been developed:

- In the vector space model [Salton 71] [Wong et al. 87], documents and queries are represented as vectors in a vector space spanned by the terms of the database. For a query-document pair, the RSV is computed by means of a similarity coefficient (e.g. dot product or cosine) of the corresponding vectors.
- In the probabilistic approach [Rijsbergen 79a] [Fuhr 88], retrieval is regarded as a stochastic process. So documents are ranked according to their probability of being relevant to the current request. It can be shown that this kind of ranking is optimum [Robertson 77]. The probability of relevance of a document is computed from probabilistic weights of the terms, which in turn are derived from relevance information about (other) query-document pairs.

In the following we describe how the probabilistic approach can be applied for retrieval of facts from databases. For a vague query, a system based on our approach first will yield an initial ranking of possible answers. Then the user is asked to give relevance judgements for some of the answers, that is, he must decide whether an answer is an acceptable solution to his problem. From this relevance feedback data, the system can derive an improved ranking of the answers for the current request. In addition to this kind of short-term learning, a major new concept of our approach is the collection of feedback data for a long-term learning process: Based on feedback information,

[1] A general discussion of the similarities and differences between information retrieval and database management systems is presented in [Eastman 89], where the issues of evaluation, matching, interaction and clustering are considered.

an improved weighting of attribute values with respect to query conditions can be derived. This way, better retrieval results for future queries can be achieved.

MODEL	CPU	MEMORY	CLOCK_RATE	DISK_SIZE	ACCESS_TIME	PRICE
A	80386	4	20	40	40	1500
B	80386	4	25	80	28	2000
C	80386	4	25	75	26	2000
D	80386	4	25	80	24	3000
E	80386	4	25	85	28	2500

Table 10.1: Example database about PCs

In order to illustrate the concepts of our approach, we will use an example of a database about personal computers throughout this chapter. The database contains information about PCs and consists of a single relation with the attributes processor type, CPU clock rate, memory size, disk size, disk access time and price (see table 10.1). Now a user may seek for a PC with an 80386 processor with a clock rate of at least 25 MHz, 4 MB of main memory and an 80 MB hard disk with an access time of less than 25 ms. Of course, he is interested in a cheap offer. This example illustrates the close relationship of this kind of problem to the field of information retrieval: It is obvious that the goal of the user, namely to select a single model for purchase, cannot be fully expressed in a query. His final decision will depend on a number of additional factors, which cannot be represented completely in the database. This means that the representation is uncertain and incomplete with respect to the application – the same situation as in text retrieval.
Now look at the sample data set in table 10.1. Here only model D fulfills all the criteria specified in the query. On the other hand, there are three more models (B, C, E) which do not fully meet the requirements, but which are significantly cheaper than model D. For this reason, the user should be informed about these models, too. It is obvious that interpreting all the criteria specified by the user as predicates in a two-valued logic would yield inappropriate results. Instead, at least some of the criteria should be regarded as vague predicates which can be fulfilled to a certain degree by attribute values.[2] This degree of fulfillment will be called indexing weights in the following.
A second kind of weighting (called query condition weighting below) refers to the different criteria specified by the user, which may not be of equal importance for him. For example, disk size may be more important than access time, so the answers should be ranked accordingly. We will describe two probabilistic retrieval models for this purpose which allow either an explicit weighting as specified by the user or an

[2]An approach based on ranking instead of Boolean logic, but using predicates from two-valued logic would e.g. not allow to distinguish between the access times of model B and C.

implicit one derived from his relevance judgements.
In addition to these two kinds of probabilistic weighting, our approach has the following features:

- Different vague predicates can be considered, like e.g. "about", "at most", "high", "low", "some".
- The approach can be applied to arbitrary data types, ranging from numbers or strings to complex objects like e.g. the shape of a geometric object.
- With the probabilistic foundation of our approach, imprecise data (e.g. disjunctive information) or missing data (i.e. null values) also can be handled easily, even for predicates from two-valued logic.

10.1.2 Foundations of the probabilistic model

Since we concentrate on the retrieval aspect of databases here, we assume a fairly simple data model, where a database consists of a set of objects to which a query may relate to; we do not consider schema transforming operations here (especially join operations), but these operations can be assumed to take place before our approach is applied, that is, the database that we regard here may be derived from the original database via some standard database operations.

Definition 1 *A database is a set of objects* $\underline{O}$*. Let* $A = \{a_1, \ldots, a_n\}$ *denote the set of attributes in the database, and* D_i *the domain for attribute* a_i*. Then the representation of a single object* $\underline{o}_m \in \underline{O}$ *is a tuple* $o_m^F =< o_m(a_1), \ldots, o_m(a_n) >$ *with* $o_m(a_i) \in D_i$*.*

As in all probabilistic models, we distinguish between an object $\underline{o}_m$ itself and its representation $o_m = o_m^F$. Although only a linear data model is assumed here, it should be emphasized that we pose no restrictions on the domains of attributes; so complex data types are allowed here, too.
Now we describe the structure of the queries. We propose our approach as an extension of database systems based on two-valued logic. A user query in such a system can be extended by a part in which one or more vague criteria can be specified.

Definition 2 *An extended query is a combination of a Boolean query and a vague query. The answer for the Boolean query is a set of objects from the database called preselected objects. The answer to the extended query is a ranked list of the preselected objects.*

In the following discussion, unless stated otherwise, we will restrict to the vague part of the query. Furthermore, no distinction between the whole database and the set of preselected objects will be made.
For our PC example, an extended query in a SQL-like notation could be specified as follows:

```
SELECT * FROM PC
   WHERE CPU    = '80386'
   AND   MEMORY = 4
```

```
RANK_BY CLOCK_RATE  >= 25,
        DISK_SIZE   >= 80,
        ACCESS_TIME <  25,
        PRICE LOW.
```

Here the WHERE-clause specifies the Boolean query which is a Boolean expression in general. The vague query starts with the RANK clause and consists of a set of conditions.

Definition 3 *A vague query is a pair $q_k = (q_k^F, q_k^J)$, where q_k^F denotes the set of vague fact conditions forming the query and q_k^J is the relevance feedback data for the query. With $\mathcal{R} = \{R, \overline{R}\}$ (relevant/nonrelevant) denoting the set of possible relevance judgements for query-object pairs, the relevance feedback data is a subset $q_k^J \subset \underline{O} \times \mathcal{R}$.*

For our PC example, the vague query formulation is

q^F = {(CLOCK_RATE, $\geq$, 25),
(DISK_SIZE, $\geq$, 80),
(ACCESS_TIME, $<$, 25),
(PRICE, low)}

For the definition of fact conditions, we first have to introduce the notion of vague predicates.

Definition 4 *A vague (or fuzzy) predicate f is either a unary predicate or a binary predicate. For each attribute $a_i \in A$ in a database, there is a set F_i^1 of unary predicates defined and a set F_i^2 of binary predicates defined.*

In contrast to fuzzy logic, a predicate is just a label here and not a mapping of attribute values onto values of a membership function. Examples for unary predicates are "low", "high", "medium" or "recent" (for dates) and also so-called fuzzy quantifiers like "some", "several", "many" . Most binary vague predicates will be vague interpretations of the standard predicates like e.g. "=", "$<$", "$\geq$".

Definition 5 *A vague fact condition c_i can have one of the three forms*

- *(a_i, f_i) with $a_i \in A$ and $f_i \in F_i^1$*
- *(a_i, f_i, d_i) with $a_i \in A$, $f_i \in F_i^2$ and $d_i \in D_i$, where d_i is called the comparison value.*
- *(a_i, f_i, a_j) with $a_i \in A$, $a_j \in A$ and $f_i \in F_i^2 \cap F_j^2$*

So a fact condition may be an attribute value with a unary attribute (e.g. PRICE low), an attribute value with a binary predicate and a comparison value (e.g. DISK_SIZE, $\geq$, 80), or two attributes compared by a binary attribute.

The goal of our probabilistic model is the estimation of the probability $P(R|q_k, o_m)$ that an object with the representation o_m will be judged relevant by a user who submitted the vague query q_k. The event space of our probabilistic model is $\underline{Q} \times \underline{O}$ where $\underline{Q}$ denotes the set of all queries submitted to the database system. A single element of this event space is a query-object pair $(\underline{q_k}, \underline{o_m})$ with a binary relevance judgement $r(\underline{q_k}, \underline{o_m}) \in \mathcal{R}$, a set q_k^F of vague fact conditions and a tupel o_m^F of attribute

values. Now we seek for an estimate of the probability $P(R|q_k, o_m)$ that an object with the representation o_m will be judged relevant with respect to query q_k. The estimation of this probability by means of a retrieval function (as a basis for the ranking of the objects) is described in section 10.1.4.
The retrieval function needs indexing weights for all the conditions in q_k^F for this purpose. For the probabilistic definition of these weights, we introduce the additional concept of correctness as an attribute of an object-condition relationship: An object may be a correct answer to a single condition, or may not. We will denote these events by the symbols C and $\overline{C}$, respectively. The decision about the correctness of an object-condition pair can be specified explicitly, that is, in addition to the relevance judgement, the user will have to judge an object with respect to each condition. However, it is also possible to derive these decisions from the relevance judgement of a query-object pair: If the pair is relevant, then the object is correct with respect to all query conditions. In the opposite case, the object is not a correct answer for any of the query conditions.
With the event of correctness defined in either of the two ways, we seek for estimates of the probability $P(C|v_i, o_m)$ that an object with representation o_m is a correct answer to the query condition v_i. The estimation of this probability is performed by the indexing task.

The indexing task consists of two steps, a description step and a decision step. In the first step, all information available about the relationship between the condition v_i and the object o_m is collected in the so-called relevance description $x(v_i, o_m)$. Based on this data, the decision step yields an estimate of the probability $P(C|x(v_i, o_m))$ that an object-condition pair described by relevance description x will be judged correct. In both steps, the different vague predicates have to be treated separately (specific relevance descriptions and estimation of indexing weights for each predicate). To keep the following explanations simple, we assume that we always regard only a single predicate in the indexing process, and that this process has to be repeated for each predicate.

Definition 6 *A relevance description $x(v_i, o_m)$ is a data structure that describes properties of the relationship between the condition v_i and the object o_m.*

Here no specific assumptions about the structure and the elements of a relevance description are necessary. However, some algorithms used in the decision step are restricted to certain types of relevance descriptions. Furthermore, it is essential that only those properties are useful as elements of the relevance description for which there is a significant correlation with the event of correctness.
This concept of relevance description yields an abstraction from specific pairs (condition, attribute value). For the binary predicates in the PC example, one could define

$$x(v_i, o_m) = \frac{o_m(a_i) - d_i}{d_i}.$$

This way, all pairs with the same relative difference between comparison value and attribute value would have identical relevance descriptions. For unary predicates, the relevance description can be defined with respect to the distribution of the attribute values in the database, e.g. the percentage of values smaller than $o_m(a_i)$.
The actual definition of the relevance description depends strongly on the application. Its elements can be values from interval scales as well as from ordinal or nominal scales (e.g. for string comparison a marker indicating whether there is a phonetic match between the two strings or not). Especially distances based on the different metrics described in [Motro 88] will provide useful information for relevance descriptions. It should be noted that a relevance description may consist of several components, that is, we can cope also with multi-dimensional metrics. This feature is important when dealing with complex data types.

In the decision step, estimates of the probabilities $P(C|x(v_i, o_m))$ are computed. For this purpose, we need a learning sample of relevance descriptions and corresponding decisions about the correctness from previous user queries.
Now, one could estimate the probability $P(C|x(v_i, o_m))$ as relative frequency from those elements of the learning sample that have the same relevance description (components of x with continuous values would have to be discretized before, see e.g. [Wong & Chiu 87]). At this point, we introduce the concept of an indexing function:
Definition 7 *Let X denote the set of relevance descriptions and $\mathbb{R}$ the set of real numbers. Then a probabilistic indexing function is a mapping $e : X \to \mathbb{R}$ such that $e(x)$ is an approximation of $P(C|x)$. We call $u(v_i, o_m) = e(x(v_i, o_m))$ the indexing weight of the object o_m with respect to condition v_i.*
As indexing function, different probabilistic classification (or learning) algorithms can be applied. The general advantage of these probabilistic algorithms over simple estimation from relative frequencies is that they yield better estimates from a learning sample given, because they use additional (plausible) assumptions about the indexing function.
Besides least square polynomials, logistic regression [Freeman 87] [Fienberg 80] seems to be well suited for this task here. This approach yields indexing functions of the form

$$e(\vec{x}) = \frac{\exp(b(\vec{x}))}{1 + \exp(b(\vec{x}))},$$

where $b(\vec{x})$ is a polynomial, and the coefficients of $b(\vec{x})$ are estimated based on the maximum likelihood method. As an example, assume that the only component of a relevance description is defined as the relative difference between d_i and $o_m(a_i)$. Now

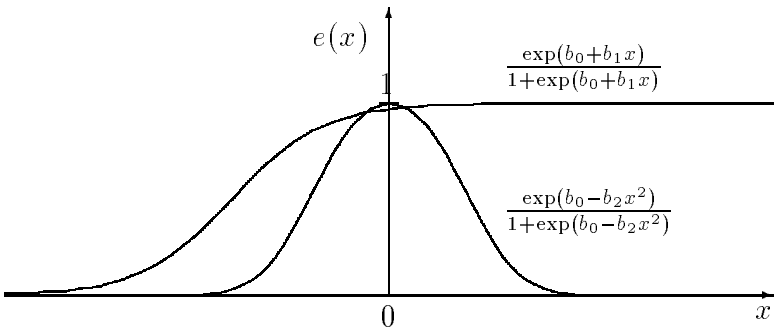


Figure 10.1: Examples of logistic indexing functions

one can define an indexing function for the vague predicate '$\geq$' as $e_1(x) = \frac{\exp(b_0+b_1x)}{1+\exp(b_0+b_1x)}$. In the case of the vague predicate '=', an appropriate indexing function would be $e_2(x) = \frac{\exp(b_0-b_2x^2)}{1+\exp(b_0-b_2x^2)}$. Both functions are illustrated in figure 10.1. As can be seen from this figure, a major advantage of logistic functions is their asymptotic behaviour. A second advantage (in comparison to polynomial functions) is related to the problem of parameter estimation: When only small learning samples are available, a prior distribution on the coefficients of $b(\vec{x})$ can be considered, thus yielding a Bayesian estimate.

As another nice property of logistic indexing functions, they support the view of vague predicates being deformations of predicates from two-valued logic: In our example, we get for $b_1 \to \infty$

$$e_1(x) = \begin{cases} 0 & \text{, if } x < 0 \\ 1-\varepsilon & \text{, if } x = 0 \\ 1 & \text{, if } x > 0 \end{cases}$$

with $\varepsilon = 1/(1+\exp(b_0))$. This result corresponds to the strict interpretation (from two-valued logic) of the predicate '$\geq$'. Similarly, we get for $b_2 \to \infty$ in $e_2(x)$ the strict interpretation of the equality predicate, namely $e_2(x) = 0$, if $x \neq 0$ and $e_2(0) = 1-\varepsilon$.

10.1.3 Indexing for missing or imprecise data

Our probabilistic indexing approach can be extended to handle also imprecise or missing data as attribute values and sets of values as comparison value. We first discuss the case of attribute values, then comparison values, and finally we show how these methods can be applied for predicates from two-valued logic, too.

Imprecise data for attribute values can be disjunctive information (e.g. assume that we only know that the clock rate of a PC model is either 25 or 33 MHz). Another major reason for imprecise attribute values is the limited precision of measurement values in technical applications. In order to handle imprecise data, a probability distribution function must be given as attribute value. For attributes with continuous values, imprecise data can be specified e.g. as an interval range or as a pair of mean and variance (for which a normal distribution is assumed). In the following we will only discuss discrete probability distributions, since the extension to continuous probability distributions is obvious.

Missing data – that is, an attribute value exists, but is not known – is often stored as null values in databases. Here also an appropriate probability distribution has to be assumed, e.g. by taking the distribution of the corresponding attribute values from similar objects or from the whole database. In most applications, an unknown attribute value does not mean that nothing at all is known about this value. Therefore, the assumption of an appropriate distribution allows to store the information actually available in the database, and to use this information in retrieval. For this reason, we regard missing values as a variant of imprecise data.

We can further extend our approach to a second interpretation of null values discussed sometimes ([Vassiliou 79] [Codd 86]), namely "attribute value not existent". For example, in an address data base, a value of the attribute "telephone number" does not exist for persons who do not have a telephone.

Now we show how these values can be handled by probabilistic indexing functions.

Definition 8 *An imprecise attribute value $o_m(a_i)$ must be specified as a discrete probability distribution over D_i, that is*

$$o_m(a_i) = \{(z_j, p_j) | z_j \in D_i and\ p_j \in [0,1]\}$$

$$with \quad \sum_{(z_j,p_j)\in o_m(a_i)} p_j = \alpha_{im}, \quad 0 \leq \alpha_{im} \leq 1.$$

This definition covers both interpretations of null values as well as the usual interpretation of imprecise data: If $\alpha_{im} = 1$, we certainly know that an attribute value exists, and with $\alpha_{im} = 0$, we represent the fact that no value exists for this attribute. In the case of $0 < \alpha_{im} < 1$, α_{im} gives the probability that an attribute value exists: For example, someone who is going to have a telephone soon gave us his number, but we are not sure if this number is valid already.

With imprecise values specified this way, their probabilistic indexing weight can be derived easily. If v_i is a condition that refers only to attribute a_i then the indexing weight is

$$P(C|v_i, o_m(a_i)) = \sum_{(z_j,p_j)\in o_m(a_i)} p_j \cdot P(C|x(v_i, z_j)).$$

As our indexing function yields approximations of the probabilistic indexing weights, we can define appropriate indexing formulas for imprecise values. In contrast to the case of precise values, we cannot show that these formulas yield optimum approximations (with respect to certain criteria). However, it is reasonable to assume that there is no significant difference between the approximation defined below and the optimum approximation.

Definition 9 *If $o_m(a_i)$ is an imprecise attribute value, then the indexing weight $u(v_i, o_m(a_i))$ is computed by the formula*

$$u(v_i, o_m(a_i)) = \sum_{(z_j, p_j) \in o_m(a_i)} p_j \cdot e(x(v_i, z_j)).$$

Our approach can be further extended to consider the disjunction or conjunction of conditions for the same attribute. Both variants are in fact syntactic elements for specifying a set of values, so we call this imprecise data as comparison value.

Definition 10 *An imprecise comparison value is a set $d_i \subseteq D_i$, where D_i is the domain of the corresponding attribute.*

The correct handling of imprecise data as comparison value depends on the specific predicate, the data type of the attribute and the type of the specification of the imprecise data (e.g. set of values vs. interval range). Two possible strategies can be applied here:

- In many cases, the indexing function for a single comparison value can be applied by object-specific selection of an appropriate value from the set or range of comparison values specified in the condition. For example, when an interval range is specified in combination with the predicate '=', then the value from the interval closest to the current attribute value is selected. As a counterexample, consider the query "List all the PC models from the manufacturer whose name is similar to 'Dandy' or 'Dundee'". For the manufacturer 'Tandy', both comparison values might yield nonzero indexing weights, and there is no simple, theoretically founded method for combining these weights.
- At a closer look, it becomes obvious that imprecise data is just another variant of a vague condition: For example, there is no systematic difference whether we specify a predicate like '>' for an attribute with continuous values or a set of constants in combination with the equality predicate for a nominal-scaled attribute. This view leads us to the development of special relevance descriptions and indexing functions for imprecise data. With this method, we can also handle rather complex specifications of imprecise data (not covered by the definition from above). As an example, assume that a user wants a PC with a green or a black and white monitor, but he prefers black and white. However, this strategy is only applicable if there is enough learning data available, that is, imprecise comparison values are used frequently in query formulations. Otherwise, one can only attempt to define appropriate indexing functions, without adaptation to the user population.

These methods for coping with imprecise data as comparison or attribute value also can be applied to predicates from two-valued logic. For this case, a binary indexing function is defined:

Definition 11 *Let p denote a predicate from two-valued logic and $d_i \subset D_i$ an imprecise comparison value. Then the indexing function is defined as*

$$e((a_i, p, d_i), o_m(a_i)) = \begin{cases} 1, & \text{if } p(z, o_m(a_i)) \text{ for} \\ & \text{any } z \in d_i \\ 0, & \text{otherwise} \end{cases}$$

For imprecise or missing attribute values, the same formulas as for vague predicates can be applied. This strategy yields the answers that a procedure for disjunctive information in two-valued logic would retrieve. For example, for the query condition $v_i = (a_i, =, \{z_1, z_2\})$ and the attribute value $o_m(a_i) = \{(z_1, \alpha), (z_2, 1-\alpha)\}$, our procedure assigns an indexing weight of 1 to the attribute value. But our approach also yields valuable results when the object is not an answer in two-valued logic: If the query condition is only $(a_i, =, z_1)$, then in general the object with attribute value $o_m(a_i) = \{(z_1, \alpha), (z_2, 1-\alpha)\}$ will be ranked ahead of all objects with null values, which in turn are ranked ahead of objects with different attribute values.

10.1.4 Retrieval functions

The task of the retrieval function $\varrho(q_k, o_m)$ is to compute relevance status values (RSVs) for query-object pairs (q_k, o_m). Then the objects can be ranked according to descending RSVs for a query. This way, a user will find the objects probably relevant at the beginning of the ranked list.

Definition 12 *A retrieval function is a mapping $\varrho : Q \times O \to I\!R$.*

Here we will discuss two different probabilistic retrieval functions, one for the initial ranking and another for an improved ranking based on relevance feedback from the user.

For the initial ranking, the user may assign weights to the conditions of his query, in order to denote their different importance w.r.t. his need. For this purpose, let w_{ik} give the weight of condition v_i in query q_k.

Based on the indexing weights $u_{im} = u(v_i, o_m)$ and the query condition weights w_{ik}, the decision-theoretic linear retrieval function [Wong & Yao 90] yields

$$\varrho_{lin}(q_k, o_m) = \sum_{v_i \in q_k^F} w_{ik} \cdot u_{im}. \tag{10.1}$$

If relevance feedback data is available, that is, the set q_k^J is not empty, the initial ranking can be further improved by applying the RPI model (see section 6.3.5.2). Let $O_k^R = \{\underline{o}_j | (\underline{o}_j, R) \in q_k^J\}$ denote the set of objects. With q_i denoting the average indexing weight for condition v_i in all objects of the database, and p_{ik} as the average

indexing weight for condition v_i in the objects judged relevant w.r.t q_k so far, the retrieval formula yields:

$$\varrho_{RPI}(q_k, o_m) = \prod_{v_i \in q_k^F} \left[\left(\frac{p_{ik}(1-q_i)}{q_i(1-p_{ik})} - 1 \right) \cdot u_{im} + 1 \right] \tag{10.2}$$

With this kind of relevance feedback, we make double use of the relevance data: The user enters relevance judgements in order to get a better ranking for his query. In addition, we collect his judgements for a long-term improvement of the system by developing probabilistic indexing functions based on this data.

10.2 Integration of Text and Fact Retrieval

10.2.1 Introduction

In the previous section, we have described the application of probabilistic IR methods for fact retrieval. Since these methods were developed originally for text retrieval, one would like to apply probabilistic retrieval for databases that contain both texts and data. In fact, in real IR applications, the objects to be retrieved hardly ever consist of text only. For example, document databases offered by public hosts also provide bibliographic information. In office information systems, documents comprise a number of attributes besides the text itself. Queries posed to these systems frequently relate to the attribute values of the documents, e.g. the name of the author (or sender) of a letter or the publication (or mailing) date. With regard to these requirements, text retrieval methods only offer a partial solution. So concepts for combining text and fact retrieval have to be devised.
In the following, we will describe briefly how the probabilistic fact retrieval approach can be extended in order to cope with objects containing text and facts.

10.2.2 Extending text retrieval methods for coping with facts

For objects containing text and facts, Definition 1 has to be extended in order to include text as well:
Definition 13 *A database is a set of objects $\underline{O}$. A single object $\underline{o}_m \in \underline{O}$ is represented by a pair $o_m = (o_m^F, o_m^T)$, where o_m^F is called the fact part and o_m^T is called the text part of $\underline{o}_m$.*
The fact part remains the same:
Definition 14 *Let $A = \{a_1, \ldots, a_n\}$ denote the set of attributes in the database, and D_i the domain for attribute a_i. Then the fact part of an object instance $\underline{o}_m$ is a tuple $o_m^F =< o_m(a_1), \ldots, o_m(a_n) >$ with $o_m(a_i) \in D_i$.*
The text part is defined in the following way:

Definition 15 *Let T denote the set of terms t_i occurring within the text parts of the objects $\underline{o}_m \in \underline{O}$. Furthermore, let Z denote the set of all possible descriptions for the occurrence of a term in a text. Then the text part of an object instance $\underline{o}_m$ is a set $o_m^T = \{(t_i, z(t_i, o_m)) | t_i \in T \wedge z(t_i, o_m) \in Z\}$ of terms and their corresponding descriptions.*

Here we make no specific assumptions about the type of terms that we are regarding; in principle, index terms can be single words, noun phrases, terms from a controlled vocabulary or even content identifiers based on some knowledge-based method. Especially for phrases, the set of all terms may not be known completely in advance. For this reason, it is not assumed here that a text is mapped directly onto a set of weighted index terms (unlike most other text indexing approaches). Instead, each term is associated with a description. The weighting process is postponed until retrieval time (at least conceptually). The details of the weighting process are described below. With this approach, terms and fact attributes are quite similar, since each term can be regarded as an attribute, and the descriptions represent the corresponding attribute values. In many applications, the text of an object will have a certain structure (e.g. title, abstract, additional keywords etc., see the example in section 10.2.3). In our probabilistic model, we do not distinguish explicitly between the different parts of a text; instead, the location information is mapped into the description of the occurrence of the term, which in turn is used for computing a weight for the term w.r.t. the text following the description-oriented indexing approach as described in [Fuhr & Buckley 91] (see below).

Queries now may relate to the fact part as well as to the text part. As an example, assume a user of an office information system seeking for all letters containing offers for office furniture that were received within the past six months. Then the extended query could be:

```
FIND DOCUMENT
  WHERE DOCTYPE='OFFER'
  RANK BY DATE > 6/91,
          TEXT: 'OFFICE FURNITURE'
```

For vague queries relating to text and facts, Definition 3 has to be changed as follows

Definition 16 *A vague query is a pair $q_k = (q_k^V, q_k^J)$, where q_k^V denotes the set of vague conditions forming the query and q_k^J is the relevance feedback data for the query. With $\mathcal{R} = \{R, \overline{R}\}$ (relevant/nonrelevant) denoting the set of possible relevance judgements for query-object pairs, the relevance feedback data is a subset $q_k^J \subset \underline{O} \times \mathcal{R}$. The (nonempty) set of vague conditions $q_k^V = q_k^F \cup q_k^T$ is the union of the set of fact conditions q_k^F and the set of text conditions q_k^T. A text condition is a term $t_i \in T$.*

In our example query, there is one fact condition (DATE > 6/91) and one text condition (TEXT: 'OFFICE FURNITURE'). In addition to the query formulation comprised of the set of vague conditions, we also consider the set of relevance feedback

judgements that we got for this query so far. This data may be exploited by the retrieval function (see below) in order to yield an improved output ranking.
In the indexing process, text and fact conditions are treated separately. For text indexing, the standard techniques can be applied.

10.2.3 An application example

In order to illustrate the concepts outlined in the previous section, we present an application of integrated text and fact retrieval here. First, the database and the indexing functions for the different attributes are described briefly, followed by the presentation of the PFTR system developed for this application.
As database, we have used a number of documents from the NTIS database which contains references to research reports of scientific projects in the USA. An example document is shown in figure 10.2.

AN 87(01):411 NTIS Order Number: AD-A172 502/7/ XAD
TI Controlling Inference. (Doctoral thesis)
AU Smith, David E.
CS Stanford Univ., CA. Dept. of Computer Science
NC Contract : N00014-81-K-0004
NR AD-A172 502/7/XAD; STAN-CS-86-1107
197 p. NTIS Prices : MF A01
Availability : Microfiche copies only.
PD 860400
LA English CY United States
OS GRA&I8701
AB Effective control of inference is a fundamental problem in Artificial Intelligence. Unguided inference leads to a combinatoral explosion of facts or subgoals for even simple domains. To overcome this problem, ...
CC 95F Bionics and artificial intelligence
CT *Artificial intelligence; Data bases; Decision making; Global; Information exchange; Problem solving; Efficiency *Inference; Control; Theses; Expert systems
UT NTISDODXA

Figure 10.2: Example NTIS document

From the different categories of these documents, the title, the summary, the controlled terms (CT) and the text of the classification codes (CC) can be accessed via text retrieval. In order to compute the indexing weights for the text part, the SMART indexing procedures have been applied ([Salton & Buckley 88]) for this specific application. These procedures have the advantage that no relevance feedback information is required for indexing. However, as shown in [Fuhr & Buckley 91], when feedback

data is available, a better retrieval quality can be achieved with probabilistic indexing methods.
For the other categories of the document as well as for the classification codes itself, vague fact retrieval methods are supplied. Except for the attribute 'date' (see below), string distance measures are used as relevance descriptions within the indexing functions of the different attributes. For these attributes, only the vague equality predicate is supported. The definition of the distance measure actually used depends on the attribute:

- For the attribute "Author" (AU), the proportion of common trigrams is computed.
- With the classification codes (CC), the length of the longest common prefix is regarded.
- For "Number of Report" (NR) and "Other Sources", the proportion of matching characters is computed.
- With the attribute "Corporate Source" (CS), the number of common words is counted.

For dates, the relative difference between the comparison value d_i and the attribute value $o_m(a_i)$ is taken as relevance description. As predicates, vague interpretations of the predicates '=', '<' and '>' are provided.
Now we describe the prototype system PFTR for probabilistic fact and text retrieval for the application presented above. This prototype serves mainly for the illustration of the user interface and the basic retrieval functions, whereas certain information system aspects (like e.g. the management of large collections of objects) have not been considered yet. The system is implemented in Smalltalk-80 similar to a Smalltalk browser; all documents are held in main memory.
Figure 10.3 shows the user interface of the PFTR system. The windows in the upper row serve for the formulation of vague attributes to which queries may relate to ("Basic Index" = text conditions). After selecting an attribute with the mouse, the next window shows the predicates that are applicable for this attribute. Having chosen a predicate, the user may enter a comparison value and a query condition weight within the two small windows below the predicate window. Then the rightmost window shows the complete vague query constructed so far. With the operation "remove", single conditions can be removed from the query formulation. When the user has completed his query formulation, he issues an "update" command which tells the system that the windows in the bottom row should be updated such that they present the answers to the query formulated in the top half. In the leftmost window of the bottom row, the ranked list of items is shown by displaying the titles. By selecting one of these titles, the complete object is shown in the rightmost window. Furthermore, a user may mark some of the objects as being relevant to his query. In turn, these titles are shown in the middle window. When the user wants to see the effect of the relevance feedback information given to the system, he again issues

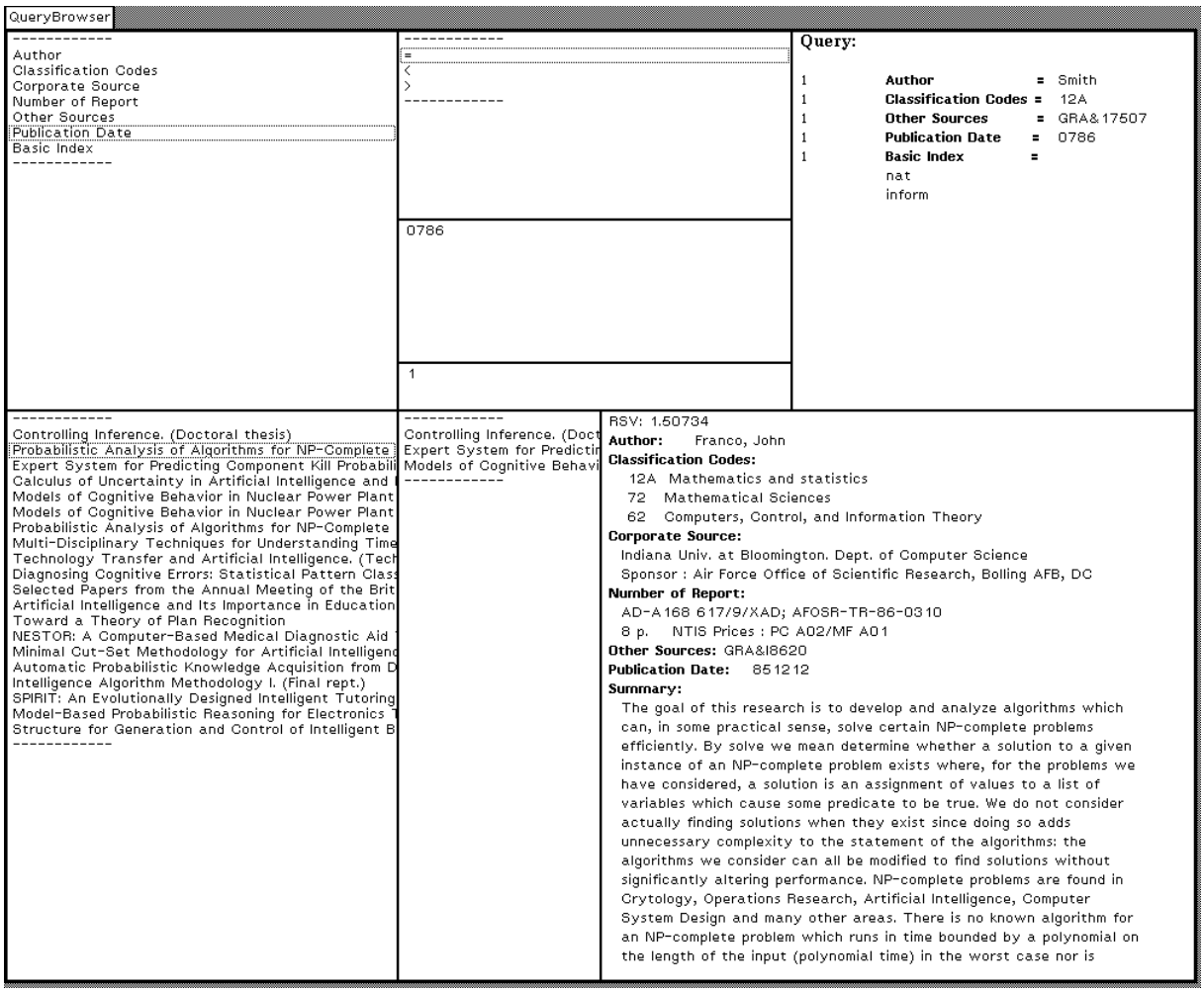


Figure 10.3: User interface of the PFTR system

the "update" command; in turn, the system computes the new ranking and displays the new list in the leftmost window. The user may also decide to modify his query by adding or removing conditions. After finishing the modification of the query, the "update" command produces the new ranking.

The basic idea behind the "update" command is to provide a generic operation for performing the retrieval step. In principle, a new ranked list of objects could be computed after each modification of the query or after marking an object as being relevant. However, an implementation of an instant re-ranking would be rather inefficient and also confusing for the user (he would have to wait for the result, and furthermore he may not want instant changes of the ranked list). For this reason, the system indicates that an update operation would be possible by displaying a grey frame around the whole PFTR browser. So the user knows that the result displayed

in the bottom half does not reflect the current state of the query and/or the feedback judgements, and that an update command would produce the latest state of the search. This strategy is rather different from that of today's systems, where user commands correspond one-to-one to system operations, and where additional commands are necessary after a retrieve command in order to show elements of the answer.

Kapitel 11

Interaktive Informationssysteme

11.1 Funktionen für interaktive Informationssysteme

Die Schnittstellen heute verfügbarer Datenbanksysteme sind nicht im Hinblick auf eine interaktive Nutzung konzipiert worden. Sinnvoll unterstützt werden lediglich Routine-Anfragen, wie sie z.B. in kaufmännischen und administrativen Anwendungen vorkommen. Um die Eignung von Datenbanksystemen für interaktive Anfragen zu verbessern, wurden benutzerfreundliche Schnittstellen entwickelt, die auf der Datenbankschnittstelle aufsetzen. Hierbei kann man nach [D'Atri & Tarantino 89] drei verschiedene Ansätze unterscheiden:

- Frageformulierungshilfen dienen dazu, die Anfrage des Benutzers in eine systemgerechte Frageformulierung zu übersetzen. Das in [Williams 84] beschriebene System RABBIT stellt die Konzepte der zugrundeliegenden Datenbank als semantisches Netzwerk dar. Anstelle der Formulierung einer Anfrage navigiert der Benutzer durch dieses Netzwerk; Bedingungen werden dadurch formuliert, daß prototypische Antwortobjekte kreiert werden, wofür spezielle Befehle zur Verfügung stehen. Der Autor nennt diesen Ansatz „query by reformulation", was den interaktiv-iterativen Prozeß der Frageformulierung unterstreicht. Das in [Fischer & Nieper-Lemke 88] vorgestellte System HELGON kombiniert diesen Ansatz mit einer wissensbasierten Terminologie-Komponente, die die Abbildung von Benutzerkonzepten auf Datenbank-Attribute unterstützt.
- Datenbank-Browser zeigen jeweils einen Ausschnitt der Datenbank in Form einer Menge von Tupeln, die gemäß einer vorgegebenen Ordnungsrelation benachbart sind. Um den angezeigten Ausschnitt auf die den Benutzer interessierenden Elemente der Datenbank zu verschieben, stehen bestimmte Navigationsoperationen zur Verfügung. In den meisten dieser Systeme erlauben aber der angebotene Befehlssatz und die vereinfachte Sicht auf die Datenbank nur die Beantwor-

tung relativ einfach strukturierter Fragen. Wie allerdings auch in [Williams 84] argumentiert wird, ist im Hinblick auf Anwendungen weniger die Mächtigkeit der Anfragesprache das entscheidende Kriterium, sondern vielmehr das Ausmaß, in dem die tatsächlich vorkommenden Frageformulierungen unterstützt werden.

- Das Konzept der Generalisierung kehrt die sonst übliche top-down-Vorgehensweise von einer Frageformulierung zu konkreten Antwortobjekten um. Stattdessen gibt der Benutzer ein Antwortobjekt vor, aus dem das System durch Generalisierung die eigentliche Anfrage ableitet. Das bekannteste Beispiel für diesen Ansatz ist das System „query by example“ ([Zloof 77]).

Neben diesen mehr an herkömmlichen Datenbankanwendungen orientierten Ansätzen wird für interaktive IR-Systeme aber eine erweiterte Anfragefunktionalität benötigt:

- Relevance Feedback ist eine Methode zur impliziten Reformulierung der Anfrage aufgrund von Benutzerbeurteilungen einzelner Antwortobjekte. Über die im Textretrieval üblichen binären Relevanzbeurteilungen hinausgehende Funktionen sollten insbesondere auch differenziertere Benutzerbeurteilungen zulassen. Das oben erwähnte System RABBIT erlaubt beispielsweise den Bezug auf einzelne Attributwerte eines Objektes, die akzeptiert oder abgelehnt werden können, und darüber hinaus können auch explizit Wertebedingungen formuliert werden.
- Um vage Anfragen zu unterstützen, muß das System einerseits entsprechende vage Prädikate als Bedingungen in der Frageformulierung zulassen, andererseits müssen auch nicht-boolesche Frageformulierungen unterstützt werden.
- Ranking von Antwortobjekten anstelle der Ausgabe einer festen Menge von Objekten ergibt sich als Konsequenz sowohl aus der systemimmanenten Unsicherheit von Daten und Repräsentationen als auch durch das Zulassen von vagen Anfragen. Durch die Ordnung der Objekte nach fallenden Wahrscheinlichkeiten $P(f_k \leftarrow o_m)$ kann der Benutzer die Liste der Objekte so weit durchsehen, wie er das für sinnvoll hält, ohne an einer bestimmten Stelle gezwungen zu sein, zur Anforderung weiterer Objekte umständlich seine Anfrage zu reformulieren (wie dies bei heutigen booleschen IR-Systemen der Fall ist).
- Automatische Strukturierung von Antwortmengen ist eine Erweiterung des Ranking-Ansatzes, bei dem im Gegensatz zur ungeordneten Antwortmenge des booleschen Retrieval eine lineare Ordnung der Objekte erzeugt wird. Gerade bei Faktendatenbanken ist es aber häufig möglich, eine differenziertere Struktur der Antwortobjekte zu erzeugen und diese dem Benutzer offenzulegen, statt ihn erst die Objekte durchsehen zu lassen, damit er selbst diese Struktur erkennt. Ein einfaches Beispiel hierfür wäre die Klasseneinteilung der Antwortmenge einer Anfrage mit einer Menge von Bedingungen entsprechend den jeweils durch die Objekte erfüllten Bedingungen.
- Das bereits oben erwähnte Konzept des Browsing bildet die zentrale Operation in Hypertext-Systemen, wo verschiedene Arten von Verknüpfungen von Objekten unterstützt werden (typisierte vs. untypisierte Kanten, globale Verknüpfun-

gen vs. Referenzen auf Objektteile wie z.B. Textpassagen oder Abbildungen). Insbesondere sollten hier auch vage Ähnlichkeitsbeziehungen zwischen Objekten unterstützt werden, und es sollte die Kombination von Browsing mit gezielten Anfragen möglich sein (z.B. eine Suche nach ähnlichen Produkten mit einem niedrigeren Preis). Eine Übersicht über die verschiedenen Arten von Navigationsoperationen in Hypertext-Systemen ist in [Krüger 91] dargestellt.

- Zooming erlaubt die Betrachtung von Objektmengen auf verschiedenen Abstraktionsniveaus. Diese Technik wird z.B. in Hypertext-Systemen angewandt, um dem Benutzer einen (globalen oder lokalen) Überblick über die Struktur des Hypertext-Dokumentes zu geben. Aber auch für Faktendatenbanken kann man sich Zooming-Operatoren vorstellen, wie dies z.B. durch die Präsentationsgraphik von Spreadsheet-Programmen eindrucksvoll demonstriert wird. Ein Beispiel für Zooming und Browsing bei komplexen Objekten ist das in [Pintado & Tsichritzis 90] beschriebene SATELLITE-System für Software-Module.
- Verzögerte Disambiguierung ist eine speziell für komplex strukturierte Datenbanken sinnvolle Technik, bei der für verschieden mögliche Interpretationen einer Anfrage zunächst deren Antworten berechnet werden, bevor ein Klärungsdialog mit dem Benutzer initiiert wird. Häufig gibt es nur für eine der möglichen Interpretationen nichtleere Antwortmengen bzw. dem Benutzer sind zusätzliche implizite Randbedingungen bekannt, die die Interpretation eindeutig machen, so daß sich die Rückfrage erübrigt.

Die vorstehend diskutierten Funktionen stellen nur die Basisfunktionen eines interaktiven Informationssystems dar. Darauf aufbauend kann dann die eigentliche Benutzerschnittstelle implementiert werden, die auf die jeweilige Anwendung zugeschnitten sein sollte. Je nach Komplexität der Anwendung ist auch eine multimodale Bedienungsoberfläche vorzusehen, die verschiedene Suchstrategien ermöglicht (siehe z.B. [Roppel & Wolf 91]). Schließlich sollten zukünftige IR-Systeme auch die Informationssuche aktiv unterstützen, indem z.B. versucht wird, Mißverständnisse zu korrigieren, Fragen überbeantwortet werden oder Reformulierungen der Anfrage vorgeschlagen werden; darüber hinaus sollte ein aktives Hilfesystem Teil der Benutzerschnittstelle sein, um die Bedienung zu erleichtern.

11.2 Zusammenspiel zwischen System und Benutzer

In ihrem Aufsatz "Where should the person stop and the information search interface start?" [Bates 90] beschäftigt sich Marcia Bates mit der Frage einer vernünftigen Aufgabenteilung zwischen System und Benutzer bei der Recherche. Im folgenden werden kurz die wesentlichen Ideen dieser Arbeit skizziert.

11.2.1 Stufen der Systembeteiligung

Level	Definition
0	**No system involvement**. All search activities human generated and executed.
1	**Displays possible activities**. System lists search activities when asked. Said activities may or may not also be executable by system (higher levels).
2	**Executes activities on command**. System executes specific actions at human command.
3	**Monitors search and recommends**. System monitors search process and recommends search activities: a) Only when searcher asks for suggestions. b) Always when it identifies a need.
4	**Executes automatically**. System executes actions automatically and then: a) Informs the searcher. b) Does not inform the searcher.

Table 11.1: Levels of system involvement

In Tabelle 11.1 werden die verschiedenen Stufen der Systembeteiligung dargestellt. Stufe 0 entspricht dabei rein manuellen Systemen. Heutige kommerzielle IR-Systeme bewegen sich meist auf Stufe 2 oder 3, da sie zum einen über primitive Hilfefunktionen verfügen, zum anderen einfache Kommandos für die Suche bereitstellen. Auf Stufe 3 bewegen sich „intelligente“ Hilfesysteme, die über einzelne Kommandos hinaus den Plan des Benutzers erkennen und ihm geeignete weitere Schritte empfehlen können. Solche Systeme werden in der Forschung teilweise untersucht [Brajnik et al. 88]. Auf der vierten Stufe finden sich schließlich die vollautomatischen Systeme, bei denen der Benutzer nur zu Beginn seinen Informationswunsch formulieren muß und dann das System den Suchprozeß kontrolliert. Aus Bates' Sicht wird in der IR-Forschung diese Art von System fast ausschließlich angestrebt; hierbei handelt es sich aber wohl eher um eine Fehlinterpretation der Arbeiten mit (probabilistischen und Vektorraum-)Modellen, die in erster Linie darauf abzielen, einen (wenn auch wesentlichen) **Teil der Systemfunktionalität** zu optimieren, wobei aber die Frage nach der Gesamtkonzeption zunächst offenbleibt.

11.2.2 Arten von Suchaktivitäten

Level	Name	Definition
1	Move	An identifiable thought or action that is a part of information searching.
2	Tactic	One or a handful of moves made to further a search.
3	Stratagem	A larger, more complex set of thoughts and/or actions than the tactic; a stratagem consists of mutiple tactics and/or moves, all designed to exploit the file structure of a particular search domain thought to contain desired information.
4	Strategy	A plan, which may contain moves, tactics, and/or stratagems, for an entire information search.

Table 11.2: Levels of search activities

Bezogen auf den Suchprozeß unterscheidet Bates die in Tabelle 11.2 dargestellten vier Arten von Suchaktivitäten, nämlich moves, Taktiken, Stratageme und Strategien. Moves sind dabei die elementaren Funktionen, wie z.B. die Eingabe eines Suchbegriffs oder der Befehl zur Anzeige eines Dokumentes. Die Granularität eines moves liegt dabei nicht von vornherein fest, sie hängt unter anderem von der jeweils betrachteten Anwendung und dem verwendeten System ab.
Die in den Tabellen 11.3 und 11.4 dargestellten Beispiele für Taktiken beziehen sich auf boolesche IR-Systeme. Es sollte aber klar sein, daß man auch für neuere Systeme Beispiele auf dieser Ebene der Suchaktivitäten finden kann.
Tabelle 11.5 zeigt Beispiele für Stratageme, die Teilbereiche der Suche, manchmal auch die ganze Suche umfassen können. Im Gegensatz dazu bezeichnet eine Strategie stets einen vorgefaßten Plan für eine ganze Recherche (*z.B. Literatur für die Vorbereitung eines Seminarvortrages über probabilistische IR-Modelle: In der Bereichsbibliothek IR-Lehrbücher suchen und entsprechende Kapitel lesen — daraus Anfrage für die Online-Recherche nach Überblicksartikeln konstruieren und diese durchführen — Überblicksartikel besorgen und darin referenzierte wichtige weitere Arbeiten beschaffen.*)

11.2.3 Kombination von Systembeteiligung und Suchaktivitäten

Im Prinzip kann nun jede Stufe der Systembeteiligung mit jeder Art der Suchaktivität kombiniert werden; allerdings ist nicht jede solche Kombination sinnvoll. Als ein Beispiel zeigt Tabelle 11.6 eine Systembeteiligung auf Stufe 1 für Taktiken (in booleschen Systemen). Heutige boolesche Systeme bewegen sich auf den Stufen 1 und 2, beschränken sich dabei aber auf moves. Aus Bates' Sicht sollte sich die gegenwärtige

MONITORING TACTICS	
CHECK	To review the original request and compare it to the current search topic to see that it is the same.
RECORD	To keep track of followed and of desirable trails not followed or not completed.
FILE STRUCTURE TACTICS	
SELECT	To break down complex search queries into subproblems and work on one problem at a time.
SURVEY	To review, at each decision point of the search, the available options before selection.
CUT	When selecting among several ways to search a given query, to choose the option that cuts out the largest part of the search domain at once .
STRETCH	To use a source for other than is intended purposes.
SEARCH FORMULATION TACTICS	
SPECIFY	To search on terms that are as specific as the information desired .
EXHAUST	To include most or all elements of the query in the initial search formulation; to add one or more of the query elements to an already-prepared search formulation.
REDUCE	To minimize the number of the elements of the query in the initial search formulation; to subtract one or more of the query elements from an already-prepared search formulation.
PARALLEL	To make the search formulation broad (or broader) by including synonyms or otherwise conceptually parallel terms.
PINPOINT	To make the search formulation precise by minimizing (or reducing) the number of parallel terms, retaining the more perfectly descriptive terms.
TERM TACTICS	
SUPER	To move upward hierarchically to a broader (superordinate) term.
SUB	To move downward hierarchically to a more specific (subordinate) term.
RELATE	To move sideways hierarchically to a coordinate term.
REARRANGE	To reverse or rearrange the words in search terms in any or reasonable orders.
CONTRARY	To search for the term logically opposite that describing the desired information.
RESPELL	To search under a different spelling.
RESPACE	To try spacing variants.

Table 11.3: Selected example search tactics

IDEA TACTICS	
RESCUE	In an otherwise unproductive approach, to check for possible productive paths still untried.
BREACH	To breach the boundaries of one's region of search, to revise one's concept of the limits of the intellectual or physical territory in which one searches to respond to a query.
FOCUS	To look at the query more narrowly, in one or both of two senses: (1) to move from the whole query to a part of it or (2) to move from a broader to a narrower conceptualization of the query.

Table 11.4: Selected example search tactics

Journal Run	Having identified a journal central to one's topic of interest, one reads or browses through issues or volumes of the journal.
Citation Search	Using a citation index or database, one starts with a citation and determines what other works have cited it.
Area Scan	After locating a subject area of interest in a classification scheme , one browses materials in the same general area.
Footnote Chase	One follows up footnotes or references, moving backward in time to other related materials.
Index or Catalog Subject Search	One looks up subject indexing terms or free text terms in a catalog or abstracting and indexing service (oneline or offline) and locates all references on one's topic of interest.

Table 11.5: Example stratagems

SEARCHER COMMAND	SYSTEM RESPONSE LIST
Too many hits	SPECIFY
	EXHAUST
	PINPOINT
	BLOCK
	SUB
Too few hits	NEIGHBOR
	TRACE
	PARALLEL
	FIX
	SUPER
	RELATE
	VARY
No hits	RESPACE
	RESPELL
	REARRANGE
	CONTRARY
	SUPER
	RELATE
	NEIGHBOR
	TRACE

SEARCHER COMMAND	SYSTEM RESPONSE LIST
Need other terms	NEIGHBOR
or	TRACE
wrong terms	SUPER
	SUB
	RELATE
Revise terms	SPACE
	RESPELL
	FIX
	REVERSE
	CONTRARY
	SUPER
	SUB
	RELATE
Revise search formulation	SPECIFY
	EXHAUST
	REDUCE
	PARALLEL
	PINPOINT
	BLOCK

Table 11.6: Tactics suggested in response to searcher request

Forschung auf die Systembeteiligung der Stufen 1–3 für Taktiken und Stratageme konzentrieren, während die Stufe 4 und / oder Strategien derzeit noch zu anspruchsvolle Aufgaben seien, um sie erfolgversprechend angehen zu können.

Literaturverzeichnis

Ammersbach, K.; Fuhr, N.; Knorz, G. (1988). Empirically Based Concepts for Materials Data Systems. In: *Proceedings of the 1988 CODATA Conference*. Karlsruhe, Germany.

Baeza-Yates, R.; Gonnet, G. (1992). A New Approach to Text Searching. *Communications of the ACM 35(10)*, S. 74–82.

Baeza-Yates, R. (1989). Algorithms for String Searching: A Survey. *SIGIR Forum 23(3/4)*, S. 34–58.

Barbara, D.; Garcia-Molina, H.; Porter, D. (1992). The Management of Probabilistic Data. *IEEE Transactions on Knowledge and Data Engineering 4(5)*, S. 487–502.

Bates, M. (1990). Where Should the Person Stop and the Information Search Interface Start? *Information Processing and Management 26(5)*, S. 575–591.

Bauer, F.; Goos, M. (1982). *Informatik*. Springer, Berlin et al.

Bertino, E.; Martino, L. (1991). Object-Oriented Database Management Systems: Concepts and Issues. *IEEE Computer 24(4)*, S. 33–47.

Biebricher, P.; Fuhr, N.; Knorz, G.; Lustig, G.; Schwantner, M. (1988). The Automatic Indexing System AIR/PHYS - from Research to Application. In: Chiaramella, Y. (Hrsg.): *11th International Conference on Research and Development in Information Retrieval*, S. 333–342. Presses Universitaires de Grenoble, Grenoble, France.

Blair, D. C. (1988). An Extended Relational Document Retrieval Model. *Information Processing and Management 24(3)*, S. 349–371.

Bookstein, A.; Swanson, D. (1974). Probabilistic Models for Automatic Indexing. *Journal of the American Society for Information Science 25*, S. 312–318.

Bookstein, A. (1983a). Information Retrieval: A Sequential Learning Process. *Journal of the American Society for Information Science 34*, S. 331–342.

Bookstein, A. (1983b). Outline of a General Probabilistic Retrieval Model. *Journal of Documentation 39(2)*, S. 63–72.

Bookstein, A. (1985). Probability and Fuzzy-Set Applications to Information Retrieval. *Annual Review of Information Science and Technology 20*, S. 117–151.

Bookstein, A.; Chiaramella, Y.; Salton, G.; Raghavan, V. (Hrsg.)(1991). *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York. ACM.

Brajnik, G.; Guida, G.; Tasso, C. (1988). IR-NLI II: Applying Man-Machine Interaction and Artificial Intelligence Concepts to Information Retrieval. In: Chiaramella, Y. (Hrsg.): *11th International Conference on Research & Development in Information Retrieval*, S. 387–399. Presses Universitaires de Grenoble, Grenoble, France.

Buckley, C.; Lewit, A. (1985). Optimization of Inverted Vector Searches. In: *Proceedings of the 8th International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 97–105. ACM, New York.

Burkart, M. (1990). Dokumentationssprachen. In: Buder, M.; Rehfeld, W.; Seeger, T. (Hrsg.): *Grundlagen der praktischen Information und Dokumentation*, S. 143–182. K.G. Saur, München et al.

Ceri, S.; Gottlob, G.; Tanca, L. (1990). *Logic Programming and Databases.* Springer, Berlin et al.

Chiaramella, Y.; Nie, J. (1990). A Retrieval Model Based on an Extended Modal Logic and its Application to the RIME Experimental Approach. In: Vidick, J.-L. (Hrsg.): *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, S. 25–44. ACM, New York.

Cleverdon, C. (1991). The Significance of the Cranfield Tests on Index Languages. In [Bookstein et al. 91], S. 3–11.

Codd, E. F. (1986). Missing Information (Applicable and Inapplicable) in Relational Databases. *SIGMOD RECORD 15(4)*, S. 53–78.

Cooper, W. (1968). Expected Search Length: A Single Measure of Retrieval Effectiveness Based on Weak Ordering Action of Retrieval Systems. *Journal of the American Society for Information Science 19*, S. 30–41.

Cooper, W. (1991). Some Inconsistencies and Misnomers in Probabilistic IR. In: Bookstein, A.; Chiaramella, Y.; Salton, G.; Raghavan, V. (Hrsg.): *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 57–61. ACM, New York.

Cox, D. (1970). *Analysis of Binary Data.* Methuen, London.

Croft, W. B.; Thompson, R. H. (1987). I3R: A New Approach to the Design of Document Retrieval Systems. *Journal of the American Society for Information Science 38(6)*, S. 389–404.

Croft, W. B. (1986). Boolean Queries and Term Dependencies in Probabilistic Retrieval Models. *Journal of the American Society for Information Science 37(2)*, S. 71–77.

Croft, W.; Turtle, H.; Lewis, D. (1991). The Use of Phrases and Structured Queries in Information Retrieval. In: Bookstein, A.; Chiaramella, Y.; Salton, G.; Raghavan, V. (Hrsg.): *Proceedings of the Fourteenth Annual International ACM*

SIGIR Conference on Research and Development in Information Retrieval, S. 32–45. ACM, New York.

Cutting, D.; Pedersen, J.; Karger, D.; Tukey, J. (1992). Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In: Belkin, N.; Ingwersen, P.; Pejtersen, M. (Hrsg.): *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 318–329. ACM, New York.

Date, C. (1990). *An Introduction to Database Systems*, Band I. Addison-Wesley, Reading (Mass.), 5th Auflage.

D'Atri, A.; Tarantino, L. (1989). From Browsing to Querying. *IEEE Data Engineering Bulletin 12(2)*, S. 46–53.

Deppisch, U. (1989). *Signaturen in Datenbanksystemen.* Dissertation, TH Darmstadt, FB Informatik.

Desai, B.; Goyal, P.; Sadri, F. (1987). Non-First Normal Form Universal Relations: An Application to Information Retrieval Systems. *Information Systems 12(1)*, S. 49–55.

Eastman, C. (1989). Approximate Retrieval: A Comparison of Information Retrieval and Database Management Systems. *IEEE Data Engineering Bulletin 12(2)*, S. 41–45.

Fagan, J. (1989). The Effectiveness of a Nonsyntactic Approach to Automatic Phrase Indexing for Document Retrieval. *Journal of the American Society for Information Science 40(2)*, S. 115–132.

Faloutsos, C. (1985). Access Methods for Text. *ACM Computing Surveys 17(1)*, S. 49–74.

Fienberg, S. (1980). *The Analysis of Cross-Classified Categorial Data.* MIT Press, Cambridge, Mass., 2. Auflage.

Fischer, G.; Nieper-Lemke, H. (1988). HELGON: Extending the Retrieval by Reformulation Paradigm. In: *Proceedings of the CHI'89*, S. 357–362. ACM, New York.

Fox, E.; Koll, M. (1988). Practical Enhanced Boolean Retrieval: Experiments with the SMART and SIRE Systems. *Information Processing and Management 24(3)*, S. 257–267.

Frakes, W.; Baeza-Yates, R. (1992). *Information Retrieval. Data Structures & Algorithms.* Prentice Hall, Englewood Cliffs.

Freeman, D. (1987). *Applied Categorial Data Analysis.* Dekker, New York.

Frei, H.; Meienberg, S.; Schäuble, P. (1991). The Perils of Interpreting Recall and Precision Values. In: Fuhr, N. (Hrsg.): *Information Retrieval*, S. 1–10. Springer, Berlin et al.

Fuhr, N.; Buckley, C. (1991). A Probabilistic Learning Approach for Document Indexing. *ACM Transactions on Information Systems 9(3)*, S. 223–248.

Fuhr, N.; Hüther, H. (1989). Optimum Probability Estimation from Empirical Distributions. *Information Processing and Management 25(5)*, S. 493–507.

Fuhr, N.; Knorz, G. (1984). Retrieval Test Evaluation of a Rule Based Automatic Indexing (AIR/PHYS). In: Van Rijsbergen, C. J. (Hrsg.): *Research and Development in Information Retrieval*, S. 391–408. Cambridge University Press, Cambridge.

Fuhr, N. (1986). Rankingexperimente mit gewichteter Indexierung. In: Deutsche Gesellschaft für Dokumentation (Hrsg.): *Deutscher Dokumentartag 1985*, S. 222–238. K.G. Saur, München, New York, London, Paris.

Fuhr, N. (1988). *Probabilistisches Indexing und Retrieval.* Dissertation, TH Darmstadt, Fachbereich Informatik. Available from: Fachinformationszentrum Karlsruhe, Eggenstein-Leopoldshafen, West Germany.

Fuhr, N. (1989a). Models for Retrieval with Probabilistic Indexing. *Information Processing and Management 25(1)*, S. 55–72.

Fuhr, N. (1989b). Optimum Polynomial Retrieval Functions Based on the Probability Ranking Principle. *ACM Transactions on Information Systems 7(3)*, S. 183–204.

Fuhr, N. (1990). A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In: McLeod, D.; Sacks-Davis, R.; Schek, H. (Hrsg.): *Proceedings of the 16th International Conference on Very Large Databases*, S. 696–707. Morgan Kaufman, Los Altos, Cal.

Fuhr, N. (1995). Modelling Hypermedia Retrieval in Datalog. In: Kuhlen, R.; Rittberger, M. (Hrsg.): *Hypertext - Information Retrieval - Multimedia, Synergieeffekte elektronischer Informationssysteme*, Band 20 von *Schriften zur Informationswissenschaft*, S. 163–174. Universitätsverlag Konstanz, Konstanz.

Fuhr, N.; Hartmann, S.; Knorz, G.; Lustig, G.; Schwantner, M.; Tzeras, K. (1991). AIR/X - a Rule-Based Multistage Indexing System for Large Subject Fields. In: *Proceedings of the RIAO'91, Barcelona, Spain, April 2-5, 1991*, S. 606–623.

Gadd, T. (1988). 'Fisching for Werds'. Phonetic Retrieval of written text in Information Retrieval Systems. *Program 22(3)*, S. 222–237.

Gonnet, G.; Baeza-Yates, R.; Snider, T. (1992). New Indices for Text: PAT Trees and PAT Arrays. In [Frakes & Baeza-Yates 92], S. 66–82.

Halpern, J. Y. (1990). An Analysis of First-Order Logics of Probability. *Artificial Intelligence 46*, S. 311–350.

Harman, D. (1988). Towards Interactive Query Expansion. In: Chiaramella, Y. (Hrsg.): *11th International Conference on Research & Development in Information Retrieval*, S. 321–331. Presses Universitaires de Grenoble, Grenoble, France.

Harman, D. (Hrsg.)(1993). *The First Text REtrieval Conference (TREC-1)*, Gaithersburg, Md. 20899. National Institute of Standards and Technology Special Publication 500-207.

Harper, D.; Walker, A. (1992). ECLAIR: an Extensible Class Library for Information Retrieval. *The Computer Journal 35(3)*, S. 256–267.

Harter, S. (1975a). A Probabilistic Approach to Automatic Keyword Indexing. Part I: On the Distribution of Speciality Words in a Technical Literature. *Journal of the American Society for Information Science 26*, S. 197–206.

Harter, S. (1975b). A Probabilistic Approach to Automatic Keyword Indexing. Part II: An Algorithm for Probabilistic Indexing. *Journal of the American Society for Information Science 26*, S. 280–289.

Hartmann, S. (1986). *Effektivitätsmaße für die Bewertung von Rankingverfahren.* Studienarbeit, TH Darmstadt, FB Informatik, Datenverwaltungssysteme II.

Heidorn, G.; Jensen, K.; Miller, L.; Byrd, R.; Chodorow, M. (1982). The Epistle Text Critiquing System. *IBM Systems Journal 21(3)*, S. 305–326.

IEEE. (1989). *IEEE Data Engineering 12(2).* Special Issue on Imprecision in Databases.

Knorz, G. (1983). *Automatisches Indexieren als Erkennen abstrakter Objekte.* Niemeyer, Tübingen.

Korfhage, R. (1997). *Information Storage and Retrieval.* Wiley, New York.

Krause, J. (1992). Intelligentes Information Retrieval. Rückblick, Bestandsaufnahme und Realisierungschancen. In: Kuhlen, R. (Hrsg.): *Experimentelles und praktisches Information Retrieval*, S. 35–58. Universitätsverlag Konstanz, Konstanz.

Krönert, G. (1988). Genormte Austauschformate für Dokumente. *(11)*, S. 71–84.

Krüger, F. (1991). Navigationsstrategien in der Informationssuche - vom Information Retrieval zu Hypertext. In: Killenberg, H.; Kuhlen, R.; Manecke, H.-J. (Hrsg.): *Wissensbasierte Informationssysteme und Informationsmanagement*, S. 104–114. Universitätsverlag Konstanz.

Kuhlen, R. (1977). *Experimentelle Morphologie in der Informationswissenschaft.* Verlag Dokumentation, München.

Kuhlen, R. (1990). Zum Stand pragmatischer Forschung in der Informationswissenschaft. In: Herget, J.; Kuhlen, R. (Hrsg.): *Pragmatische Aspekte beim Entwurf und Betrieb von Informationssystemen. Proceedings des 1. Internationalen Symposiums für Informationswissenschaft*, S. 13–18. Universitätsverlag Konstanz, Konstanz.

Kuhlen, R. (1991). Zur Theorie informationeller Mehrwerte. In: Killenberg, H.; Kuhlen, R.; Manecke, H.-J. (Hrsg.): *Wissensbasierte Informationssysteme und Informationsmanagement*, S. 26–39. Universitätsverlag Konstanz.

Kwok, K. (1990). Experiments with a Component Theory of Probabilistic Information Retrieval Based on Single Terms as Document Components. *ACM Transactions on Information Systems 8*, S. 363–386.

Kwok, K. (1991). Query Modification and Expansion in a Network with Adaptive Architecture. In: Bookstein, A.; Chiaramella, Y.; Salton, G.; Raghavan, V. (Hrsg.): *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 192–201. ACM, New York.

Lee, S. (1992). An Extended Relational Database Model For Uncertain And Imprecise Information. In: *Proceedings of the 18th VLDB Conference*, S. 211–220. Morgan Kaufman.

Lenat, D.; Guha, R.; Pittman, K. (1990). Cyc: Toward Programs With Common Sense. *Communications of the ACM 33(8)*, S. 30–49.

Losee, R. (1988). Parameter Estimation for Probabilistic Document-Retrieval Models. *Journal of the American Society for Information Science 39(1)*, S. 8–16.

Lucarella, D. (1988). A Search Strategy for Large Document Bases. *Electronic Publishing 1(2)*, S. 105–116.

Lustig, G. (Hrsg.) (1986). *Automatische Indexierung zwischen Forschung und Anwendung.* Olms, Hildesheim.

Lynch, C. (1991). Nonmaterialized Relations and the Support of Information Retrieval Application by Relational Database Systems. *Journal of the American Society for Information Science 42(6)*, S. 389–398.

Macleod, I. (1991). Text Retrieval and the Relational Model. *Journal of the American Society for Information Science 42(3)*, S. 155–165.

Margulis, E. (1991). *N-Poisson Document Modelling Revisited.* Technical Report 166, ETH Zürich, Departement Informatik, Institut für Informationssysteme.

Maron, M.; Kuhns, J. (1960). On Relevance, Probabilistic Indexing, and Information Retrieval. *Journal of the ACM 7*, S. 216–244.

Meghini, C.; Rabitti, F.; Thanos, C. (1991). Conceptual Modeling of Multimedia Documents. *IEEE Computer 24(10)*, S. 23–30.

Meghini, C.; Sebastiani, F.; Straccia, U.; Thanos, C. (1993). A Model of Information Retrieval Based on a Terminological Logic. In: Korfhage, R.; Rasmussen, E.; Willett, P. (Hrsg.): *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 298–308. ACM, New York.

Motro, A. (1988). VAGUE: A User Interface to Relational Databases that Permits Vague Queries. *ACM Transactions on Office Information Systems 6(3)*, S. 187–214.

Motro, A. (1990). Accommodating Imprecision in Database Systems: Issues and Solutions. *Sigmod record 19(4)*, S. 69.

Motzkin, D. (1991). An Efficient Directory System for Information Retrieval. In [Bookstein et al. 91], S. 291–304.

Nie, J. (1989). An Information Retrieval Model Based on Modal Logic. *Information processing & management. 25(5)*, S. 477–491.

Nilsson, N. J. (1986). Probabilistic Logic. *Artificial Intelligence 28*, S. 71–87.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufman, San Mateo, Cal.

Pintado, X.; Tsichritzis, D. (1990). SaTellite: A Navigation Tool for Hypermedia. In: *Proceedings of the Conference on Office Information Systems.* ACM, New York.

Prade, H.; Testemale, C. (1984). Generalizing Database Relational Algebra for the Treatment of Incomplete/Uncertain Information and Vague Queries. *Information Science 34*, S. 115–143.

Rabitti, F.; Savino, P. (1990). Retrieval of Multimedia Documents by Imprecise Query Specification. In: Bancilhon, F.; Thanos, C.; Tsichritzis, D. (Hrsg.): *Advances in Database Technology - EDBT '90*, S. 203–218. Springer, Berlin et al.

Rabitti, F.; Savino, P. (1991). Image Query Processing based on Multi-Level Signatures. In: Bookstein, A.; Chiaramella, Y.; Salton, G.; Raghavan, V. (Hrsg.): *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 305–314. ACM, New York.

Rabitti, R.; Zezula, P. a. (1990). A Dynamic Signature Technique for Multimedia Databases. In: Vidick, J.-L. (Hrsg.): *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, S. 193–210. ACM, New York.

Raghavan, V.; Wong, S. (1986). A Critical Analysis of Vector Space Model for Information Retrieval. *Journal of the American Society for Information Science 37(5)*, S. 279–286.

Raghavan, V.; Bollmann, P.; Jung, G. (1989). A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. *ACM Transactions on Information Systems 7(3)*, S. 205–229.

van Rijsbergen, C. J.; Sparck Jones, K. (1973). A Test for the Separation of Relevant and Non-relevant Documents in Experimental Retrieval Collections. *Journal of Documentation 29*, S. 251–257.

van Rijsbergen, C. J. (1977). A Theoretical Basis for the Use of Co-Occurrence Data in Information Retrieval. *Journal of Documentation 33*, S. 106–119.

van Rijsbergen, C. J. (1979a). *Information Retrieval.* Butterworths, London, 2. Auflage.

van Rijsbergen, C. J. (1979b). *Information Retrieval*, chapter 6, S. 111–143. Butterworths, London, 2. Auflage.

van Rijsbergen, C. J. (1986). A Non-Classical Logic for Information Retrieval. *The Computer Journal 29(6)*, S. 481–485.

van Rijsbergen, C. J. (1989). Towards an Information Logic. In: Belkin, N.; van Rijsbergen, C. J. (Hrsg.): *Proceedings of the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 77–86. ACM, New York.

van Rijsbergen, C. J.; Harper, D. J.; F., P. M. (1981). The Selection of Good Search Terms. *Information Processing and Management 17*, S. 77–91.

Roberts, C. (1979). Patial-Match Retrieval via the Method of Superimposed Coding. *Proceedings of the IEEE 67(12)*, S. 1624–1642.

Robertson, S.; Harding, P. (1984). Probabilistic Automatic Indexing by Learning from Human Indexers. *Journal of Documentation 40*, S. 264–270.

Robertson, S.; Sparck Jones, K. (1976). Relevance Weighting of Search Terms. *Journal of the American Society for Information Science 27*, S. 129–146.

Robertson, S. (1977). The Probability Ranking Principle in IR. *Journal of Documentation 33*, S. 294–304.

Robertson, S. (1986). On Relevance Weight Estimation and Query Expansion. *Journal of Documentation 42*, S. 182–188.

Robertson, S.; Maron, M.; Cooper, W. (1982). Probability of Relevance: A Unification of Two Competing Models for Document Retrieval. *Information Technology: Research and Development 1*, S. 1–21.

Rocchio, J. (1966). *Document Retrieval Systems - Optimization and Evaluation.* Report ISR-10 to the NSF, Computation Laboratory, Harvard University.

Rölleke, T. (1994). *Equivalences of the Probabilistic Relational Algebra.* Technical report, University of Dortmund, Department of Computer Science.

Roppel, S.; Wolf, C. (1991). Graphische Recherche und intelligente Hilfe als Komponenten des multimodalen Werkstoffinformationssystems WING. In: Killenberg, H.; Kuhlen, R.; Manecke, H.-J. (Hrsg.): *Wissensbasierte Informationssysteme und Informationsmanagement*, S. 115–124. Universitätsverlag Konstanz.

Salton, G.; Buckley, C. (1988). Term Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management 24(5)*, S. 513–523.

Salton, G.; Buckley, C. (1990). Improving Retrieval Performance by Relevance Feedback. *Journal of the American Society for Information Science 41(4)*, S. 288–297.

Salton, G.; McGill, M. J. (1983). *Introduction to Modern Information Retrieval.* McGraw-Hill, New York.

Salton, G.; Smith, M. (1989). On the Application of Syntactic Methodologies in Automatic Text Analysis. In: Belkin, N.; van Rijsbergen, C. J. (Hrsg.): *Proceedings of the Twelfth Annual International ACMSIGIR Conference on Research and Development in Information Retrieval*, S. 137–150. ACM, New York.

Salton, G. (Hrsg.) (1971). *The SMART Retrieval System - Experiments in Automatic Document Processing.* Prentice Hall, Englewood, Cliffs, New Jersey.

Salton, G. (1986). Another Look at Automatic Text-Retrieval Systems. *Communications of the ACM 29(7)*, S. 648–656.

Salton, G. (1988). Automatic Text Indexing Using Complex Identifiers. In: *Proceedings of ACM Conference on Document Processing Systems (December 5-9, 1988, Santa Fe, New Mexico)*, S. 135–144. ACM, New York.

Salton, G.; Fox, E.; Wu, H. (1983). Extended Boolean Information Retrieval. *Communications of the ACM 26*, S. 1022–1036.

Schek, H.-J.; Pistor, P. (1982). Data Structures for an Integrated Database Management and Information Retrieval System. In: *Proceedings of the 8th International Conference on Very Large Data Bases*, S. 197–207. Morgan Kaufman, Los Altos, Cal.

Schmidt, M.; Pfeifer, U. (1993). Eignung von Signaturbäumen für Best-Match-Anfragen. In: *Proceedings 1. GI-Fachtagung Information Retrieval*, S. 125–138. Universitätsverlag Konstanz, Konstanz.

Schneider, R.; Kriegel, H.-P.; Seeger, B.; Heep, S. (1989). Geometry-Based Similarity Retrieval of Rotational Parts. In: *Proceedings 2nd International Conference on Data and Knowledge Systems for Manufacturing and Engineering.*

Schürmann, J. (1977). *Polynomklassifikatoren für die Zeichenerkennung. Ansatz, Adaption, Anwendung.* Oldenbourg, München, Wien.

Sembok, T.; van Rijsbergen, C. J. (1990). SILOL: A simple Logical-Linguistic Document Retrieval System. *Information Processing and Management 26(1)*, S. 111–134.

Sparck Jones, K.; Willet, P. (1997). *Readings in Information Retrieval.* Morgan Kaufman, Palo Alto.

Srinivasan, P. (1990). On the Generalization of the Two-Poisson Model. *Journal of the American Society for Information Science 41(1)*, S. 61–66.

Staud, J. (1990). Statistische Information. In: Buder, M.; Rehfeld, W.; Seeger, T. (Hrsg.): *Grundlagen der praktischen Information und Dokumentation*, S. 402–427. K.G. Saur, München et al.

Stirling, K. (1975). The Effect of Document Ranking on Retrieval System Performance: A Search for an Optimal Ranking Rule. In: *Proceedings of the American Society for Information Science 12*, S. 105–106.

Turtle, H.; Croft, W. B. (1990). Inference Networks for Document Retrieval. In: Vidick, J.-L. (Hrsg.): *Proceedings of the 13th International Conference on Research and Development in Information Retrieval*, S. 1–24. ACM, New York.

Turtle, H.; Croft, W. (1991). Evaluation of an Inference Network-Based Retrieval Model. *ACM Transactions on Information Systems 9(3)*, S. 187–222.

Ullman, J. (1988). *Principles of Database and Knowledge-Base Systems*, Band I. Computer Science Press, Rockville (Md.).

Vassiliou, Y. (1979). Null Values in Database Management - a Denotational Semantics Approach. In: *Proceedings of the ACM SIGMOD International Conference on the Management of Data.* ACM, New York.

Verhoeff, J.; Goffmann, W.; Belzer, J. (1961). Inefficiency of the Use of Boolean Functions for Information Retrieval Systems. *Communications of the ACM 4*, S. 557–558.

Westbrook, J.; Rumble, J. (Hrsg.)(1983). *Computerized Materials Data Systems*, Gaithersburg, Maryland 20899, USA. National Bureau of Standards.

Willett, P. (1988). Recent Trends in Hierarchic Document Clustering: A Critical Review. *Information Processing and Management 24(5)*, S. 577–597.

Williams, M. (1984). What makes RABBIT run? *International Journal on Man-Machine Studies 21*, S. 333–352.

Witten, I.; Moffat, A.; Bell, T. (1994). *Managing Gigabytes : Compressing and Indexing Documents and Images*. Van Nostrand Reinhold.

Wong, A.; Chiu, D. (1987). Synthesizing Statistical Knowledge from Incomplete Mixed-Mode Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence 9(6)*, S. 796–805.

Wong, S.; Yao, Y. (1989). A Probability Distribution Model for Information Retrieval. *Information Processing and Management 25(1)*, S. 39–53.

Wong, S.; Yao, Y. (1990). Query Formulation in Linear Retrieval Models. *Journal of the American Society for Information Science 41(5)*, S. 334–341.

Wong, S.; Ziarko, W.; Raghavan, V.; Wong, P. (1987). On Modeling of Information Retrieval Concepts in Vector Spaces. *ACM Transactions on Database Systems 12(2)*, S. 299–321.

Yu, C.; Buckley, C.; Lam, K.; Salton, G. (1983). A Generalized Term Dependence Model in Information Retrieval. *Information Technology: Research and Development 2*, S. 129–154.

Zadeh, L. (1965). Fuzzy Sets. *Information and Control 8*, S. 338–353.

Zimmermann, H. (1991). Ein Verfahren zur automatischen Trunkierung beim Zugang zu textbezogenen Informationsbanken. In: Killenberg, H.; Kuhlen, R.; Manecke, H.-J. (Hrsg.): *Wissensbasierte Informationssysteme und Informationsmanagement*, S. 125–144. Universitätsverlag Konstanz.

Zloof, M. (1977). Query-by-Example: A Data Base Language. *IBM Systems Journal 16(4)*, S. 324–343.