

Information Retrieval
Skriptum zur Vorlesung im SS 06

Norbert Fuhr

7. Dezember 2006

Inhaltsverzeichnis

1	Einführung	5
1.1	Was ist Information Retrieval?	5
2	IR-Konzepte	7
2.1	Daten — Information — Wissen	7
2.2	Konzeptionelles Modell für IR-Systeme	8
3	Evaluierung	11
3.1	Effizienz und Effektivität	12
3.2	Relevanz	12
3.3	Distributionen	13
3.4	Standpunkte und Bewertungsmaße	13
3.4.1	Benutzerstandpunkte	14
3.4.2	Benutzer- vs. Systemstandpunkte	14
3.5	Maße für boolesches Retrieval	14
3.5.1	Recall, Precision und Fallout	14
3.5.2	Recall-Abschätzung	15
3.5.3	Frageweise Vergleiche	16
3.5.4	Mittelwertbildung	17
3.6	Rangordnungen	18
3.6.1	Lineare Ordnung	19
3.6.2	Schwache Ordnung	19
3.7	Interpretation von Recall-Precision-Graphen	22
3.7.1	Abbruchkriterium: Anzahl der relevanten Dokumente (NR)	23
3.7.2	Abbruchkriterium: Anzahl der Dokumente	25
3.8	Mittelwertbildung und Signifikanztests bei Rangordnungen	26
3.9	Evaluierungsinitiativen: TREC, CLEF, NTCIR und INEX	26
3.9.1	Evaluierungsmaße	27
3.9.2	TREC: Text REtrieval Conference	28
3.9.3	CLEF: Cross-Language Evaluation Forum	28
3.9.4	NTCIR: NACSIS Test Collection Project	28
3.9.5	INEX: Initiative for the Evaluation of XML Retrieval	28
3.10	Evaluierung von interaktivem Retrieval	29
4	Wissensrepräsentation für Texte	30
4.1	Problemstellung	30
4.2	Freitextsuche	30
4.2.1	Informatischer Ansatz	31
4.2.2	Computerlinguistischer Ansatz	32
4.3	Dokumentationssprachen	38
4.3.1	Allgemeine Eigenschaften	38
4.3.2	Klassifikationen	38
4.3.3	Thesauri	44

4.3.4	RDF (Resource Description Framework)	48
4.3.5	Dokumentationssprachen vs. Freitext	50
4.4	Beurteilung der Verfahren zur Repräsentation von Textinhalten	51
4.5	Zusammenhang zwischen Modellen und Repräsentationen	52
4.5.1	Textrepräsentation für IR-Modelle	52
4.5.2	Einfache statistische Modelle	52
5	Nicht-probabilistische IR-Modelle	53
5.1	Notationen	53
5.2	Überblick über die Modelle	54
5.3	Boolesches Retrieval	54
5.3.1	Mächtigkeit der booleschen Anfragesprache	55
5.3.2	Nachteile des booleschen Retrieval	55
5.4	Fuzzy-Retrieval	55
5.4.1	Beurteilung des Fuzzy-Retrieval	57
5.5	Das Vektorraummodell	57
5.5.1	Coordination Level Match	58
5.5.2	Relevance Feedback	58
5.5.3	Dokumentindexierung	61
5.5.4	Beurteilung des VRM	62
5.6	Dokumenten-Clustering	62
5.6.1	Cluster-Retrieval	63
5.6.2	Ähnlichkeitssuche von Dokumenten	64
5.6.3	Probabilistisches Clustering	64
5.6.4	Cluster-Browsing	66
5.6.5	Scatter/Gather-Browsing	67
6	Probabilistic Models in Information Retrieval	69
6.1	Introduction	69
6.2	Basic concepts of relevance models	69
6.2.1	The binary independence retrieval model	69
6.2.2	A conceptual model for IR	71
6.2.3	Parameter learning in IR	72
6.2.4	Event space	73
6.2.5	The Probability Ranking Principle	74
6.3	Some relevance models	77
6.3.1	A description-oriented approach for retrieval functions	77
6.3.2	The binary independence indexing model	81
6.3.3	A description-oriented indexing approach	82
6.3.4	The 2-Poisson model	84
6.3.5	Retrieval with probabilistic indexing	84
6.4	IR as uncertain inference	86
6.5	Parameter estimation	88
6.5.1	Parameter estimation and IR models	88
6.5.2	Standard methods of parameter estimation	88
6.5.3	Optimum parameter estimation	92
7	Models based on propositional logic	94
7.1	A Probabilistic Inference Model	94
7.2	Classical IR models	95
7.2.1	Disjoint basic concepts	95
7.2.2	Nondisjoint basic concepts	97

8	Models based on predicate logic	105
8.1	Introduction	105
8.2	Terminological logic	105
8.2.1	Thesauri	105
8.2.2	Elements of terminological logic	106
8.2.3	Semantics of MIRTL	108
8.2.4	Retrieval with terminological logic	108
8.3	Datalog	111
8.3.1	Introduction	111
8.3.2	Hypertext structure	111
8.3.3	Aggregation	112
8.3.4	Object hierarchy	112
8.3.5	Retrieval with terminological knowledge	113
8.4	Probabilistic Datalog	115
8.4.1	Introduction	115
8.4.2	Informal description of Datalog _P	115
8.4.3	Syntax	116
8.4.4	Semantics of Datalog _P	117
8.4.5	Evaluation of Datalog _P programs	119
8.4.6	Datalog _P with independence assumptions	120
8.4.7	Further application examples	121
8.4.8	Probabilistic rules	122
9	IR-Systeme	125
9.1	Ebenenarchitektur	125
9.2	Konzeptionelle Ebene	125
9.2.1	Stufen der Systembeteiligung	126
9.2.2	Arten von Suchaktivitäten	126
9.2.3	Kombination von Systembeteiligung und Suchaktivitäten	127
9.3	Semantic level	128
9.3.1	The FERMI multimedia retrieval model	128
9.3.2	POOL: A probabilistic object-oriented logic	130
9.3.3	FMM and POOL	131
10	Implementierung von IR-Systemen	132
10.1	Hardware-Aspekte	132
10.1.1	Speichermedien	132
10.1.2	Ein-/Ausgabegeräte	133
10.1.3	Kommunikationsnetzwerke	133
10.2	Aufbau von IRS	133
10.2.1	Funktionale Sicht	133
10.2.2	Dateistruktur	133
10.2.3	Dialogfunktionen herkömmlicher IRS	135
10.3	Dokumentarchitekturen	135
10.3.1	ODA	135
10.3.2	Markup-Sprachen	139
10.4	Zugriffspfade	144
10.4.1	Scanning	144
10.4.2	Ähnlichkeit von Zeichenketten	152
10.4.3	Invertierte Listen	154
10.4.4	Signaturen	156
10.4.5	PAT-Bäume	168

Vorwort

Das vorliegende Skript dient als Begleitlektüre zur vierstündigen Vorlesung „Information Retrieval“.

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 und Norbert Gövert gestaltet. Ein Teil des Abschnitts über Signaturen basiert auf einer Vorlage von Renato Vinga. Tung Huynh, Thorsten Ernst und Michael Günnewig 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 früherer Fassungen dieses Skriptums, die zur Korrektur zahlreicher Fehler führten.

Literatur

Leider gibt es kaum geeignete Lehrbücher zum Gebiet des Information Retrieval. Das einzig empfehlenswerte deutschsprachige IR-Buch ist [Ferber 03]. Thematisch breiter und m.E. das derzeit beste Buch zu diesem Thema ist [Baeza-Yates & Ribeiro-Neto 99], auch wenn dessen einzelne Kapitel von unterschiedlicher Qualität sind. Eine stärker kognitiv orientierte Sichtweise als in dieser Vorlesung liegt dem Buch [Belew 00] zugrunde, das ansonsten aber sehr empfehlenswert ist. Das Werk [Agosti et al. 01] präsentiert die Kurse einer europäischen IR-Sommerschule und ist insbesondere wegen der Zusammenstellung der unterschiedlichen Facetten des Gebietes lesenswert. 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 jährliche ACM-CIKM-Konferenz (International Conference on Information and Knowledge Management) sowie die alle drei Jahre stattfindende RIAO-Tagung (Recherche d'informations assisté par Ordinateur) befassen 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*.

Kapitel 1

Einführung

Will man den Gegenstand des Information Retrieval (IR) 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.

Das klassische Anwendungsgebiet des IR sind Literaturdatenbanken, die heute in Form Digitaler Bibliotheken zunehmend an Bedeutung gewinnen. IR ist besonders populär geworden durch die Anwendung in Internet-Suchmaschinen; dadurch kommt jeder Internet-Nutzer mit IR-Methoden in Berührung. Neben der Suche in Texten werden auch zunehmend IR-Anwendungen für multimediale Daten realisiert, wobei insbesondere Bildretrieval-Methoden eine gewisse Verbreitung erfahren haben.

Jeder, der eine dieser Anwendungen wiederholt genutzt hat, wird die wesentlichen Unterschiede zwischen IR-Anwendungen und denen klassischer Datenbanksysteme leicht erkennen:

- Die Formulierung einer zum aktuellen Informationsbedürfnis passenden Anfrage bereitet erhebliche Probleme.
- Meistens durchläuft der Prozess der Anfrageformulierung mehrere Iterationen, bis passende Antworten gefunden werden.
- Anfragen liefern potentiell sehr viele Antworten (vgl. die Gesamtzahl der Treffer bei Internet-Suchmaschinen), aber nur wenige davon sind für den Nutzer interessant.
- Das vorgenannte Problem entschärft sich durch die vom System bereitgestellte Rangordnung der Antworten, wodurch potentiell relevante Antworten gehäuft am Anfang der Rangliste auftauchen (z.B. betrachten bei Internet-Suchmaschinen mehr als 90% aller Nutzer nur die ersten 10 Antworten)
- Bei Textdokumenten, aber noch stärker bei Bildern zeigt sich, dass die systemintern verwendete Repräsentation des Inhalts von Dokumenten teilweise inadäquat, auf jeden Fall aber mit Unsicherheit behaftet ist.

1.1 Was ist Information Retrieval?

Zur Definition des Gebietes legen wir hier die Beschreibung der Aufgaben und Ziele der Fachgruppe „Information Retrieval“ innerhalb der „Gesellschaft für Informatik“ zugrunde:

„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.

Oftmals wird IR auch eingeschränkt auf die inhaltsorientierte Suche in (multimedialen) Dokumenten betrachtet. (Tatsächlich behandeln wir in diesem Skriptum fast ausschließlich Modelle und Methoden aus diesem Bereich.) Für diese Art der Suche kann man folgende Abstraktionsstufen unterscheiden:

Syntax: Hierbei wird ein Dokument als Folge von Symbolen aufgefasst. Methoden, die auf dieser Ebene operieren, sind z.B. die Zeichenkettensuche in Texten sowie die Bildretrievalverfahren, die nach Merkmalen wie Farbe, Textur und Kontur suchen.

Semantik beschäftigt sich mit der Bedeutung eines Dokumentes. Methoden zur Repräsentation der Semantik eines Textes haben eine lange Tradition im Bereich der Wissensrepräsentation; semantisches Bildretrieval müsste die Suche nach Bildern unterstützen, die z.B. bestimmte (Klassen von) Objekten enthalten (Menschen, Häuser, Autos, . . .).

Pragmatik orientiert sich an der Nutzung eines Dokumentes für einen bestimmten Zweck. Zum Beispiel sucht ein Student Literatur zur einem vorgegebenen Seminarthema. Bildarchive werden häufig von Journalisten in Anspruch genommen, um einen Artikel zu illustrieren; dabei ist meist das Thema vorgegeben, aber nicht der semantische Bildinhalt.

Generell lässt sich festhalten, dass Nutzer meistens an einer Suche auf der pragmatischen Ebene interessiert sind. Insbesondere bei nicht-textuellen Dokumenten können dies heutige IR-Systeme aber kaum leisten.

Abschließend zu diesen Betrachtungen geben wir hier die in [Rijsbergen 79a] skizzierten Dimensionen des IR wieder (Tabelle 1.1). Dabei steht die mittlere Spalte für mehr Datenbank-orientierte Anwendungen, während die rechte Spalte eher klassische IR-Anwendungen charakterisiert. Allerdings kann man die jeweiligen Einträge einer Zeile auch als die zwei Endpunkte einer kontinuierlichen Skala auffassen, auf der es viele mögliche Zwischenlösungen gibt. Solche Lösungen sind insbesondere bei Anwendungen, mit semistrukturierten Daten (z.B. XML) gefragt.

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

Tabelle 1.1: Dimensionen des Information Retrieval

Kapitel 2

IR-Konzepte

2.1 Daten — Information — Wissen

Datenbanksysteme enthalten Daten. IR-Systeme sollen die Suche nach Information¹ 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 aber leider im Gegensatz zur sonst in der Informatik verwendeten steht. Daher verwenden wir hier die allgemein übliche Begrifflichkeit, allerdings in Kombination mit den Erläuterungen aus der Informationswissenschaft (siehe Abbildung 2.1). 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 Information. Dementsprechend enthalten also Datenbanksysteme nicht nur Daten, sondern auch Information, weil zusätzlich zu den Daten zumindest ein Teil der Semantik des jeweiligen Anwendungsgebietes auch im System modelliert wird. Genauso enthält jedes IR-System Information (im Gegensatz etwa zu dem Fall, wo man Texte einfach in einer Datei abspeichert und mit Hilfe eines Texteditors durchsucht).

Wissen schließlich ist auf der pragmatischen Ebene definiert. In Abwandlung von [Kuhlen 90] lässt sich dies so formulieren: „Wissen ist die Teilmenge von Information, die von jemandem in einer konkreten Situation zur Lösung von Problemen benötigt wird“. Da dieses Wissen häufig nicht vorhanden ist, wird danach in externen Quellen gesucht. Hierbei dient ein Informationssystem dazu, aus der gespeicherten Information das benötigte Wissen zu extrahieren. Wir sprechen auch von Informationsflut, wenn uns große

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

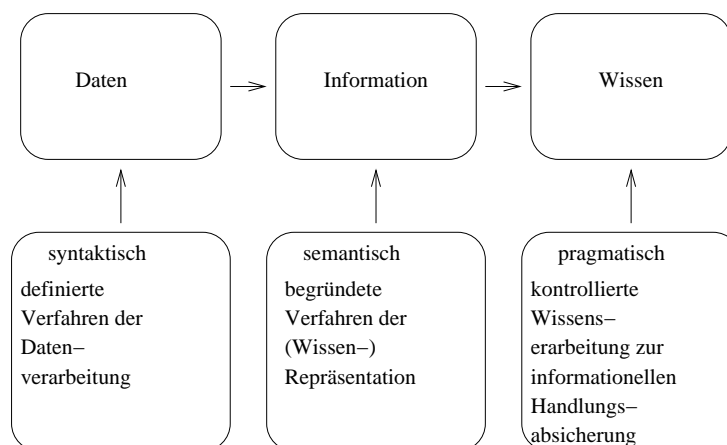


Abbildung 2.1: Daten — Information — Wissen

Mengen an Information zugeleitet werden, aus denen wir nur mit Mühe das benötigte Wissen extrahieren können. Daher sind wir auch bereit, für gezielt bereitgestelltes Wissen zu zahlen (z.B. Tageszeitung, werbefreies Fernsehen). Somit kann man die Transformation von Information in Wissen als einen Mehrwert erzeugenden Prozess sehen [Kuhlen 91]. Schlagwortartig lässt sich die Beziehung zwischen Information und Wissen ausdrücken durch die Formulierung „Wissen ist Information in Aktion“.

Als anschauliches Beispiel kann man hierzu die online verfügbaren UNIX-Manuals betrachten. Diese beinhalten Information 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 das in diesem Fall benötigte Wissen gerade die entsprechende Teilmenge der insgesamt in den Manuals verfügbaren, umfangreichen Information. Da nur ein geringer Teil der gesamten Information benötigt wird, besteht der Mehrwert des Wissens (so sie durch die hierzu verfügbaren Werkzeuge wie z.B. das man-Kommando geliefert wird) gerade in ihrer gezielten Bereitstellung.

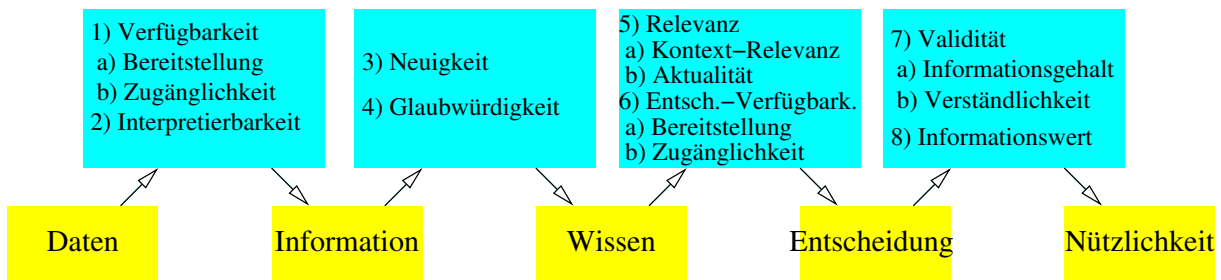


Abbildung 2.2: Wissen zur Entscheidungsunterstützung

Wie oben erwähnt, dient Wissen zur „informationellen Handlungsabsicherung“. Im Kern geht es dabei um Rolle des Wissens zur Entscheidungsunterstützung. Dieser Zusammenhang wird durch Abbildung 2.2 verdeutlicht. Hier sind zugleich einige Qualitätskriterien für die einzelnen Transformationsschritte angegeben. Diese Aspekte werden vornehmlich im Bereich des Wissensmanagement betrachtet, wo die Rolle des Wissens innerhalb organisatorischer Strukturen und bei Arbeitsabläufen untersucht wird.

2.2 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 der ursprünglich im Electronic Publishing entwickelten Trennung zwischen Layout und logischer Struktur ist. Die wesentliche Idee dieses Dokumentmodells besteht darin, dass es mehrere Sichten auf den Inhalt eines Dokumentes gibt (siehe Abb. 2.3)

- 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 (oder inhaltliche) Sicht bezieht sich auf die Semantik des Inhalts eines Dokumentes. Für IR-Systeme ist diese Sicht essentiell, da ansonsten nur primitive Suchoperationen in der Form der Zeichenkettensuche möglich wären.

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

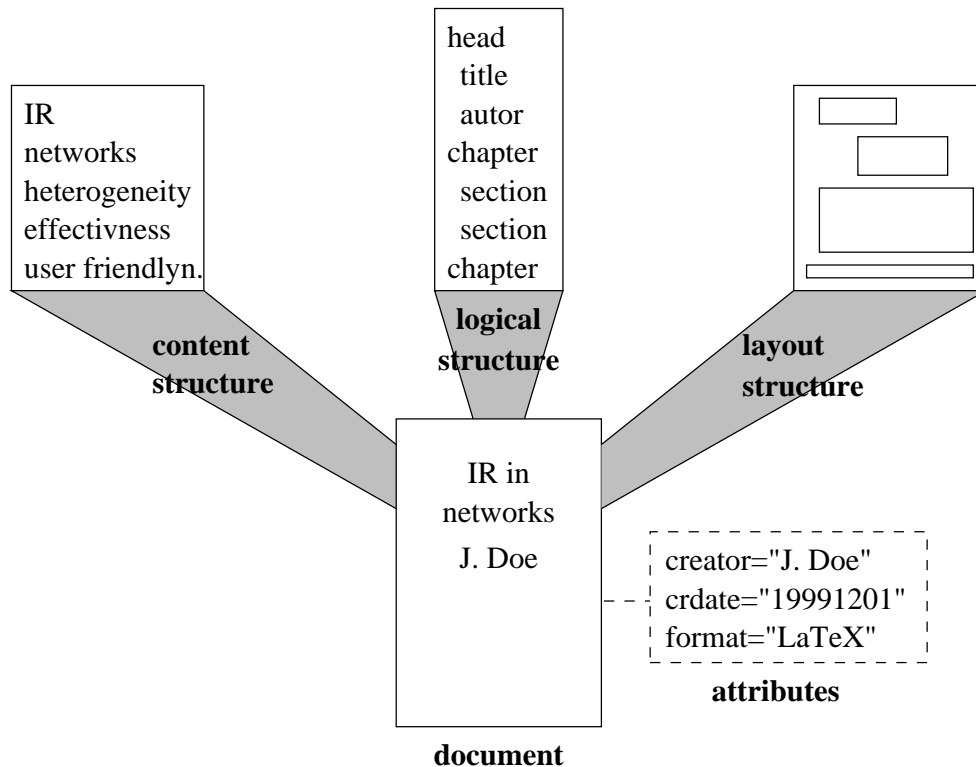


Abbildung 2.3: Sichten auf Dokumente

diese bei den vorherrschenden 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 2.4 dargestellte konzeptionelle Modell formulieren. Jedes Dokument einer Datenbasis wird mittels geeigneter Erschließungsverfahren eine entsprechende interne Darstellung (die wir hier Repräsentation nennen wollen) transformiert, in der neben Struktur und Layout auch der Inhalt geeignet repräsentiert wird. Diese wird zu einer Dokument-Beschreibung verdichtet, die all jene Angaben zum Dokument enthält, nach denen gesucht werden kann. Eine Anfrage des Benutzers enthält zunächst einmal Selektionsbedingungen, die sich auf alle drei Aspekte eines Dokumentes beziehen können.

Beispiele für Anfragen mit Bezug zu den verschiedenen Arten von Sichten 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 Aspekte in Kombination auftreten.

Die resultierende Antwortdokumente werden dann projiziert, um geeignete Surrogate zu erzeugen; diese Projektion kann sich wiederum auf alle drei Aspekte eines Dokumentes beziehen; z.B. würden Titel und fragebezogene Zusammenfassung eine Kombination aus strukturellen und inhaltlichen Aspekten darstellen, ein Thumbnail der ersten Seite wäre eine Layout-bezogene Projektion. Die projizierten Dokumente/Surrogate werden dann als Menge dargestellt. Auch hier können Inhalt, Struktur und Layout der Darstellung gewählt werden: Im einfachsten Fall werden die Surrogate als lineare Liste (nach fallenden Retrievalwerten oder syntaktischen Merkmalen wie z.B. Erscheinungsjahr geordnet) dargestellt; sie können aber auch nach inhaltlichen Kriterien geclustert oder klassifiziert werden, oder als Zusammenfassung von mehreren Dokumenten dargestellt werden. Anspruchsvollere Visualisierungen nutzen die Layout-Möglichkeiten stärker aus.

Anhand dieser Abbildung kann auch der Aspekt der Unsicherheit verdeutlicht werden. Die Ableitung der Repräsentation aus dem eigentlichen Dokument ist eine wesentliche Quelle von Unsicherheit. Speziell bei Texten oder multimedialen Dokumenten kann deren Inhalt nur unzureichend erschlossen werden

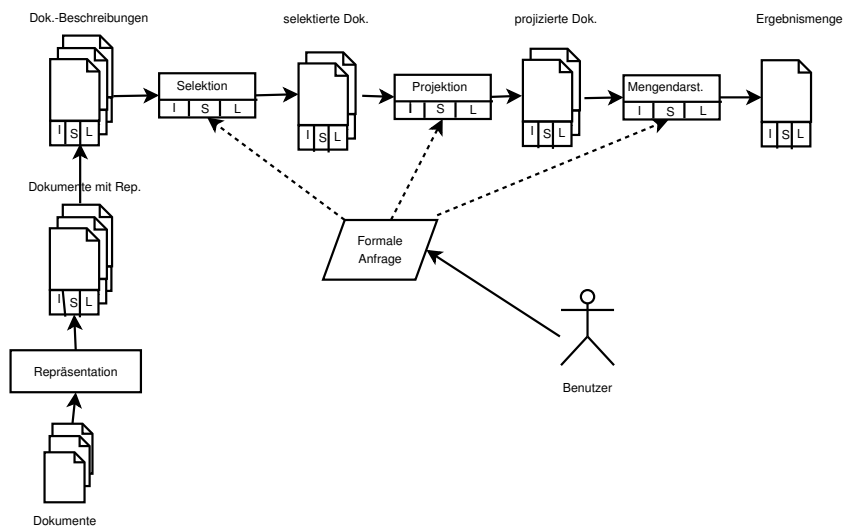


Abbildung 2.4: Konzeptionelles Modell für Informationssysteme

Auf der Seite der Fragen ergeben sich die gleichen Probleme der Unsicherheit, insbesondere bei der Abbildung des Informationswunsches auf die formale Anfrage. Zusätzlich spielt hier das für IR-Anwendungen typische Moment der Vagheit eine wichtige Rolle. 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 wenig geeignet. Zudem ist die Forschungsliteratur im IR reich an 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.

Evaluierungen sollen die Qualität eines Systems beurteilen helfen. Dabei muss man berücksichtigen, dass es unterschiedliche Blickwinkel auf ein IRS gibt, z.B. die von Benutzern, Käufern, Managern, Herstellern oder Entwicklern. Für jede dieser Gruppen sind bestimmte Aspekte eines Systems wichtiger als andere, stehen andere Fragen bei der Evaluierung im Vordergrund. Einige dieser Fragen könnten etwa sein:

- Was kann ich ändern, um die Qualität eines Systems zu verbessern?
- Welche Art der Textrepräsentation ist am besten?
- Welches Retrievalmodell liefert die besten Ergebnisse?
- Welche Qualität weist ein System auf?
- Welches System ist besser?
- Welches System soll ich kaufen?
- Wie kann ich Qualität messen?
- Was bedeutet Qualität für mich?

Um diese Fragen zu beantworten, können jeweils geeignete Evaluierungen konzipiert und durchgeführt werden. Generell sollte jede Evaluierung — insbesondere wenn sie wissenschaftlichen Maßstäben genügen will — folgende zwei Eigenschaften erfüllen:

Zuverlässigkeit Dieselbe Untersuchung im gleichen Kontext sollte stets dieselben Ergebnissen liefern, das Experiment sollte also *wiederholbar* sein.

Validität Die Beobachtungen sollten mit den 'tatsächlichen' Verhältnissen übereinstimmen, um die *Gültigkeit* der Ergebnisse zu gewährleisten.

Dabei ist zu beachten dass IR-Experimente stets stochastische Experimente sind, dass also bei Wiederholungen eines Experimentes sich in der Regel nicht genau die gleichen Messwerte wie beim vorherigen Versuch ergeben. Daher muss eine ausreichende Zahl von Versuchen durchgeführt werden (z.B. eine größere Menge von Anfragen betrachtet werden), um sowohl Zuverlässigkeit als auch Validität zu erreichen.

Abhängig von der Entwicklungsphase des zu untersuchenden Systems kann man folgende Arten von Evaluierungen unterscheiden:

- Die *Formative Evaluierung* wird zu Beginn der Systementwicklung durchgeführt. Abhängig vom anvisierten Anwendungskontext und weiteren Projektzielen werden u.a. Funktionalität, Zielen und gewünschte Ergebnisse des zu entwickelnden Systems festgelegt. Hierbei handelt es sich also nicht um Evaluierung im engeren Sinne, sondern eher um eine Anforderungsspezifikation.
- Die *iterative Evaluierung* wird begleitend zur Systemeentwicklung durchgeführt und dient im wesentlichen als Grundlage für bzw. zur Verifizierung von Basis Entwurfsentscheidungen.
- Demgegenüber steht die *summative Evaluierung* am Projektende, die das realisierte System mit den Projektzielen vergleicht.

- Die *komparative Evaluierung* vergleicht mehrere Systeme (bzw. -Komponenten), meist auf der Basis standardisierter Qualitätsmaße

Im Folgenden werden wir hauptsächlich Methoden für die iterative und die komparative Evaluierung betrachten.

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 häufig 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).

Bei Evaluierungen hält man meistens einen der beiden Parameter (Kosten, Nutzen) konstant. Bei der Evaluierung interaktiver Systeme wird oft der Nutzen konstant gehalten, indem man Fragen verwendet, auf die es eine eindeutige Antwort gibt (z. B. Wie lange ist der Rhein?); dann misst man den hierfür benötigten Aufwand des Benutzers. Alternativ kann man den Aufwand fest vorgeben: z. B. wird bei interaktiven Systemen den Nutzern eine feste Zeit zur Suche vorgegeben; meist nimmt man aber an, dass der Benutzer nur die Anfrage formuliert und keine weitere Interaktion stattfindet. Wir werden die folgenden Betrachtungen allein auf den Fall konstanten Aufwands beschränken — nicht zuletzt deshalb, weil die hier betrachteten Verfahren keine oder nur wenige, standardisierte Interaktionsmöglichkeiten zwischen Benutzer und System zulassen. Daher genügt es, allein die resultierende Qualität eines Informationssystems zu betrachten.

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.

In der Literatur werden meist 4 Arten von Relevanz unterschieden:

Situative Relevanz beschreibt die (tatsächliche) Nützlichkeit des Dokumentes in Bezug auf die Aufgabe, aus der heraus das Informationsbedürfnis entstanden ist. Diese Auffassung von Relevanz orientiert sich also an unserer Definition des Informationsbegriffs. Allerdings kann man die situative Relevanz praktisch kaum erfassen, es handelt sich also eher um ein theoretisches Konstrukt.

Pertinenz ist die subjektiv vom Benutzer empfundene Nützlichkeit des Dokumentes in Bezug auf das Informationsbedürfnis. Wenn also der Anfragende selbst Relevanzurteile abgibt, so handelt es sich genau genommen um Pertinenzurteile.

Objektive Relevanz ist die von einem oder mehreren neutralen Beobachtern beurteilte Beziehung zwischen dem geäußerten Informationswunsch und dem Dokument. Der Relevanzbegriff wird häufig bei Systemevaluierungen zugrunde gelegt.

Systemrelevanz bezeichnet die von einem automatischen System geschätzte Relevanz des Dokumentes in Bezug auf die formale Anfrage. In diesem Skript verwenden wir hierfür die Bezeichnung Retrievalwert (englisch: *Retrieval Status Value (RSV)*), der durch die so genannte Retrievalfunktion berechnet wird.

Im Folgenden wird zwischen Pertinenz und objektiver Relevanz nicht mehr unterschieden. Zudem machen wir die Einschränkung, dass die Relevanzskala zweistufig ist, also aus den beiden Werten „relevant“ und „nicht relevant“ besteht.

3.3 Distributionen

Distributionen sind abstrakte Darstellungen 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 und 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:

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

Die einzelnen Ränge werden dabei durch „|“ getrennt, „+“ bezeichnet ein relevantes und „-“ ein nichtrelevantes Dokument.

3.4 Standpunkte und Bewertungsmaße

Jedem Bewertungsmaß liegt ein bestimmter Standpunkt bezüglich 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 Abschnitt ?? das Nützlichkeitsmaß vorgestellt.

3.5 Maße für boolesches Retrieval

3.5.1 Recall, Precision und Fallout

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 (Abbildung 3.1).

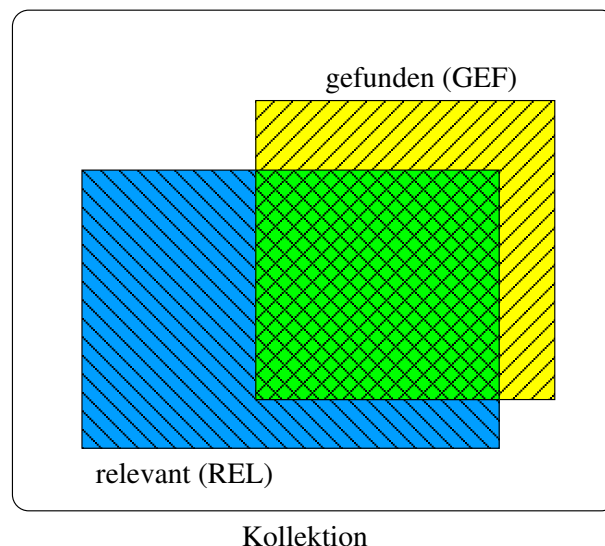


Abbildung 3.1: Mengen der relevanten und gefundenen Dokumente

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

$$\begin{aligned} \text{Precision:} \quad p &:= \frac{|REL \cap GEF|}{|GEF|} \\ \text{Recall:} \quad r &:= \frac{|REL \cap GEF|}{|REL|} \\ \text{Fallout:} \quad f &:= \frac{|GEF - REL|}{|ALL - REL|} \end{aligned}$$

Hierbei gibt die 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 Fähigkeit des Systems bewertet, irrelevante Dokumente vom Benutzer fernzuhalten.

Da es sich bei Retrievalexperimenten um stochastische Experimente handelt, sollte man die Messwerte auch entsprechend interpretieren. Im Falle der Precision $p = |REL \cap GEF|/|GEF|$ wird damit die Wahrscheinlichkeit approximiert, dass ein (zufällig ausgewähltes) gefundenes Dokument relevant ist. Analog schätzt man mit dem Recall $r = |REL \cap GEF|/|REL|$ die Wahrscheinlichkeit, dass ein (zufällig ausgewähltes) relevantes Dokument gefunden wird. Entsprechendes gilt für den Fallout. Diese probabilistische Interpretation der Retrievalmaße spielt eine wesentliche Rolle bei den Optimalitätsbetrachtungen zum probabilistischen Ranking-Prinzip.

Für konkrete Anwendungen — insbesondere solche, bei denen anstelle eines Booleschen Retrieval eine Rangliste von Antworten zum System geliefert wird — werden häufig Varianten dieser Maße verwendet, die an den jeweiligen Kontext angepasst wurden.

- Beim Web-Retrieval kann man davon ausgehen, dass die meisten Benutzer (nach empirischen Untersuchungen ca. 90 %) sich nur die erste Seite der Ergebnisliste anschauen, die in der Regel 10 Antworten enthält. Ein passendes Maß ist daher die Precision nach 10 Dokumenten, die meist als 'Prec@10' bezeichnet wird. Ein extremer (aber z.B. von Google unterstützter) Standpunkt wäre die Precision des ersten Dokumentes (Prec@1).
- Bei Evaluierungsinitiativen wie TREC, CLEF oder INEX werden in analoger Weise z.B. Prec@5, Prec@10, Prec@30 und Prec@100 parallel betrachtet, um Benutzerklassen zu simulieren, die sich jeweils die entsprechende Anzahl Dokumente anschauen. Ergänzend werden Recall-orientierte Standpunkte simuliert, indem die Precision-Werte bei bestimmten Recall-Punkten gemessen werden, wobei dann zusätzlich über mehrere Recall-Punkte (entsprechend verschiedenen Benutzerklassen) gemittelt wird.

3.5.2 Recall-Abschätzung

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* zu bestimmen. Folgende Näherungsmethoden wurden hierzu vorgeschlagen:

Vollständige Relevanzbeurteilung: einer repräsentativen Stichprobe der gesamten Datenbasis: Da *REL* sehr viel kleiner als die gesamte Datenbasis ist (z. B. mögen 100 von 10^7 Dokumenten relevant sein), müsste die repräsentative Stichprobe schon einen relativ großen Teil der Datenbasis umfassen, was zuviel Beurteilungsaufwand erfordert.

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.

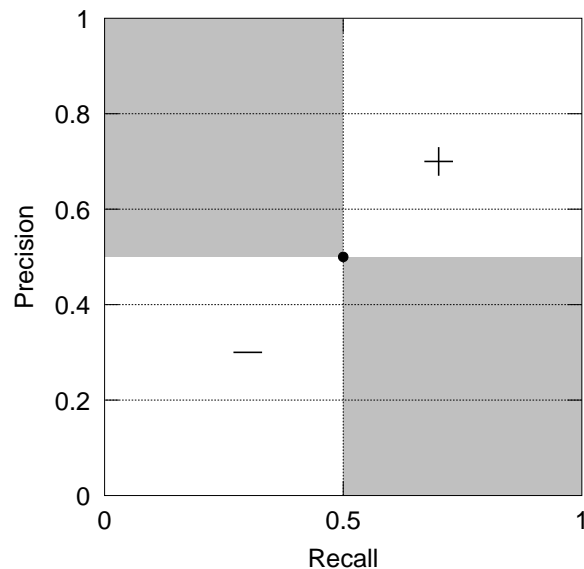


Abbildung 3.2: Darstellung eines Retrievalergebnisses als Punkt im Recall-Precision-Graphen

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.

Ableich 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.

Pooling-Methode: (Retrieval mit mehreren Systemen): Man wendet mehrere IR-Systeme auf denselben Dokumentenbestand an und mischt die Ergebnisse verschiedener Systeme zu den gleichen Anfragen. In der Regel gibt es starke Überlappungen in den Antwortmengen der verschiedenen Systeme, so dass der Aufwand nicht linear mit der Anzahl betrachteter Systeme wächst [Harman 95]. Dieses Verfahren wird derzeit beim Vergleich experimenteller Systeme im Rahmen der TREC- und CLEF-Konferenzen angewandt (Abschnitt 3.9).

Außer den ersten beiden Verfahren liefern alle Methoden nur untere Schranken für *REL*; die gemessenen Recall-Werte sind daher im Allgemeinen zu optimistisch.

3.5.3 Frageweise Vergleiche

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 3.2 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 3.2).

Als eine gängige Methode, (r, p) -Paare durch eine einzige Zahl auszudrücken, hat sich das *F*-Maß

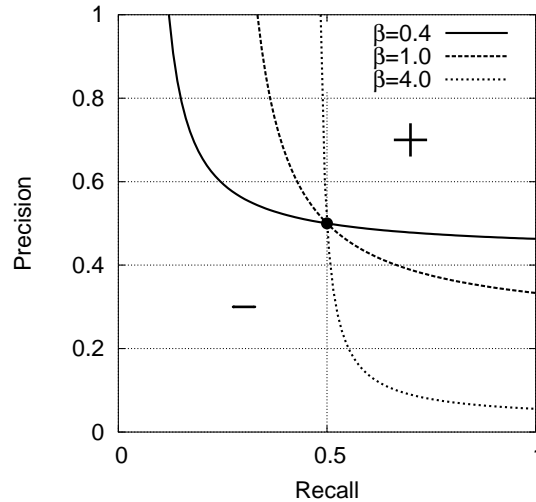


Abbildung 3.3: Aufteilung von Recall-Precision-Punkten durch das F -Maß: Für $F = 0.5$ und verschiedene β -Werte finden sich bessere Recall-Precision-Punkte im rechten oberen Bereich.

durchgesetzt. Abhängig von einem zu wählenden Parameter β berechnet sich dieses Maß zu

$$F_\beta = \frac{(\beta^2 + 1) \cdot p \cdot r}{\beta^2 \cdot p + r}$$

Hierbei gibt β die relative Wichtung des Recall an ($\beta = 0$: nur Precision zählt; $\beta = \infty$: nur Recall zählt). Üblicherweise setzt man $\beta = 1$, arbeitet also mit dem F_1 -Maß. Abbildung 3.3 zeigt die Aufteilung von Recall-Precision-Punkten in bessere und schlechtere Ergebnisse durch das F -Maß: Bezogen auf den F -Wert 0,5 für verschiedene β -Werte finden sich bessere Recall-Precision-Punkte jeweils im rechten oberen Bereich, schlechtere Punkte auf der jeweils anderen Seite der Kurven.

Als Alternative zu diesen kombinierten Maßen kann man auch Kostenmaße betrachten; diese werden insbesondere bei Systemen zur Informationsfilterung häufig eingesetzt. Dabei geht man von folgender Kontingenztafel aus und zählt die Anzahl Dokumente h_i für jeden der vier Fälle:

	relevant	irrelevant
gefunden	h_g^R	h_g^I
nicht gefunden	h_n^R	h_n^I

Die allgemeine Formel für die Gesamtkosten ergibt sich dann als gewichtete Summe der verschiedenen Anzahlen:

$$C = C_g^R \cdot h_g^R + C_g^I \cdot h_g^I + C_n^R \cdot h_n^R + C_n^I \cdot h_n^I$$

Dabei sind C_g^R , C_g^I , C_n^R und C_n^I die Kostenparameter für die vier Fälle. Im einfachsten Fall könnte man etwa wählen $C_g^R = C_n^I = 0$ und $C_g^I = C_n^R = 1$. $C_g^I = C_n^R = 1$.

Will man dagegen ein System zur Filterung von Spam-E-mails bewerten, so sollte zwar das System möglichst viele 'relevante' (d.h. Spam-Mails) identifizieren, aber möglichst keine 'irrelevanten' (nicht-Spam) Mails selektieren. Um also h_g^I (im Vergleich h_n^R , der Anzahl an den Benutzer weitergeleiteten Spam-Mails) zu möglichst klein zu halten, sollten also entsprechende Werte $C_g^I \gg C_n^R$ gewählt werden.

3.5.4 Mittelwertbildung

Wie oben erwähnt, 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 $i = \{1, \dots, N\}$ 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:

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

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 unter Umständen behoben werden.

Aus stochastischer Sicht approximiert die Makro-Methode den Erwartungswert für die Precision zu einer zufällig ausgewählten Anfrage. Somit 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:

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

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. Aus stochastischer Sicht wird hier die Wahrscheinlichkeit approximiert, dass ein (zufällig ausgewähltes) gefundenes Dokument aus einer der N Anfragen relevant ist.

Analoge Betrachtungen gelten für Recall und Fallout.

Ein spezielles 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. Seien $\Delta_1 = (+-)$ und $\Delta_2 = (+ + - - -)$ Retrievalergebnisse, zu denen später das Retrievalergebnis $\Delta = (+ + - - - - -)$ hinzugefügt wird.

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

3.6 Rangordnungen

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

Bei Rangordnungen muss man zusätzlich unterscheiden, ob eine lineare (totale) Ordnung der Dokumente aus der Datenbasis vorliegt oder nur eine schwache Ordnung (es können mehrere Dokumente im selben Rang sein).

Wir stellen Retrievalergebnisse durch das in Abschnitt 3.3 beschriebene Schema dar. Die Distributionen Δ_1 und Δ_2 dienen im Folgenden als Beispiele für lineare Rangordnungen:

$$\begin{aligned} \Delta_1 &= (+|+|-|+|-|+|-|-|-|-|-|-|+|-) \\ \Delta_2 &= (+|-|+|+|+|-|-|-|-|-|-|+|-|+|-) \end{aligned}$$

Die zugrundegelegte fiktive Dokumentkollektion enthält also 14 Dokumente, von denen im Fall von Δ_1 5 und im Fall von Δ_2 6 Dokumente als relevant beurteilt wurden.

Die Distributionen Δ_3 und Δ_4 finden als Beispiele für schwache Rangordnungen Verwendung:

n	Dokumentnr.	$\times = \text{rel.}$	Recall	Precision
1	588	\times	0.2	1.00
2	589	\times	0.4	1.00
3	576		0.4	0.67
4	590	\times	0.6	0.75
5	986		0.6	0.60
6	592	\times	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	\times	1.0	0.38
14	990		1.0	0.36

Tabelle 3.1: Recall und Precision für Δ_1 nach dem Nachweis von n Dokumenten bei linearer Ordnung

$$\begin{aligned} \Delta_3 &= (+ + - | + + + + - - - - - | + + | + - - - - | + - (80)) \\ \Delta_4 &= (+ | + - | + + + + + - - - - - - - - | + + | + + - (80)) \end{aligned}$$

Hierbei enthält die Dokumentenkollektion insgesamt 100 Dokumente (der letzte Rang enthält jeweils 80 nicht relevante Dokumente), von denen bei Δ_3 10 und bei Δ_4 11 Dokumente als relevant beurteilt wurden.

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_1 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 3.4 (links) gezeigte Bild. Um die Übersichtlichkeit zu erhöhen, kann man die einzelnen Punkte mit Geradenstücken verbinden (lineare Interpolation, Abbildung 3.4 rechts). 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 Qualitätsmaße für mehrere Rangordnungen in einem einzigen Graphen darstellen möchte (siehe Abbildungen 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 die zu den Distributionen Δ_3 und Δ_4 gehörigen Recall- und Precision-Werte für die einzelnen Ränge, die in Abbildung 3.7 graphisch dargestellt sind.

Ein möglicher Weg zur Definition von eindeutigen Precision-Werten für vorgegebene Recall-Werte besteht in der sogenannten $PRECALL_{ceiling}$ -Methode. Diese Methode ist an Saltons Interpolationsmethode angelehnt. Hierbei wird zunächst die Precision an jedem einfachen Recall-Punkt berechnet. Bei n relevanten

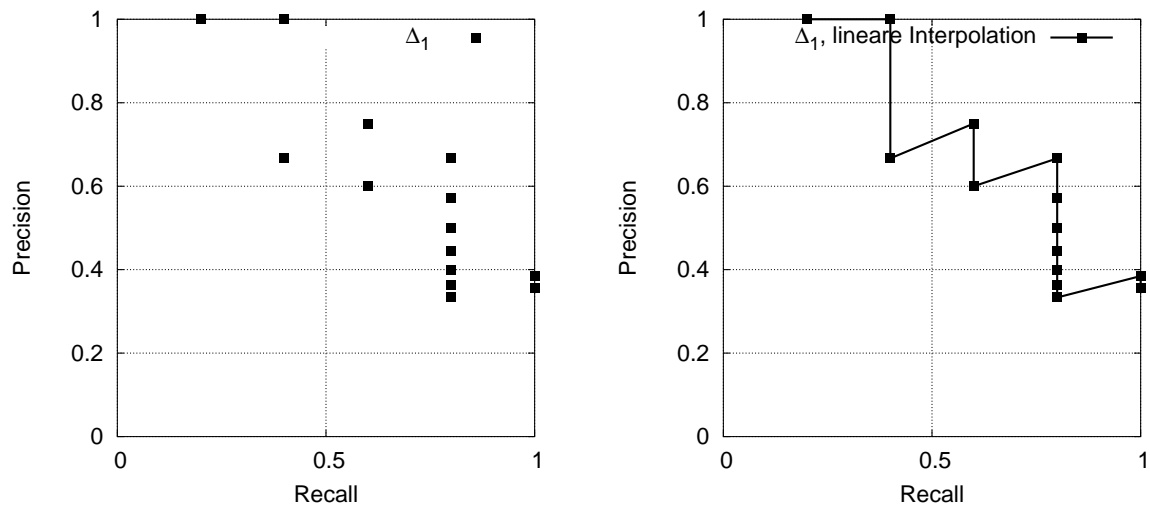


Abbildung 3.4: Graphische Darstellung der Werte aus Tabelle 3.1 (Δ_1), rechts mit linearer Interpolation der Punkte.

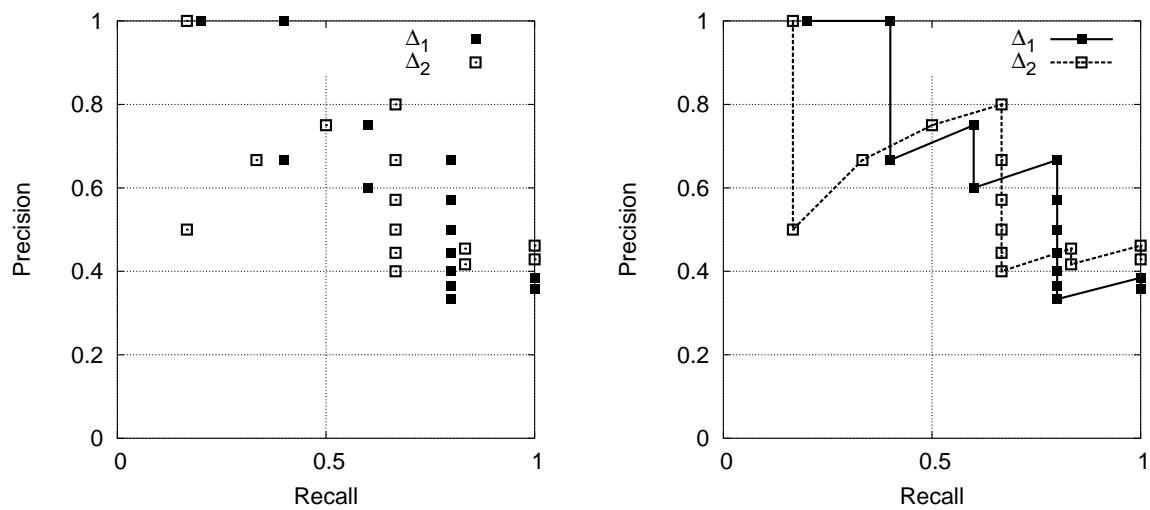


Abbildung 3.5: Graphische Darstellung der Werte für zwei verschiedene Rangordnungen (Δ_1 und Δ_2), rechts mit linearer Interpolation der Punkte.

Rang	Δ_3		Δ_4	
	Recall	Precision	Recall	Precision
1	0.20	0.67	0.09	1.00
2	0.60	0.50	0.18	0.67
3	0.80	0.57	0.63	0.44
4	0.90	0.47	0.81	0.50
5	1.00	0.10	1.00	0.11

Tabelle 3.2: Rangabhängige Recall- und Precision-Werte zu Δ_3 und Δ_4

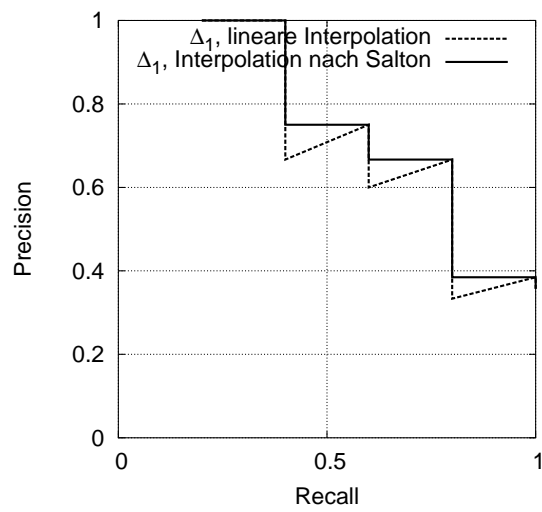


Abbildung 3.6: Interpolation nach Salton

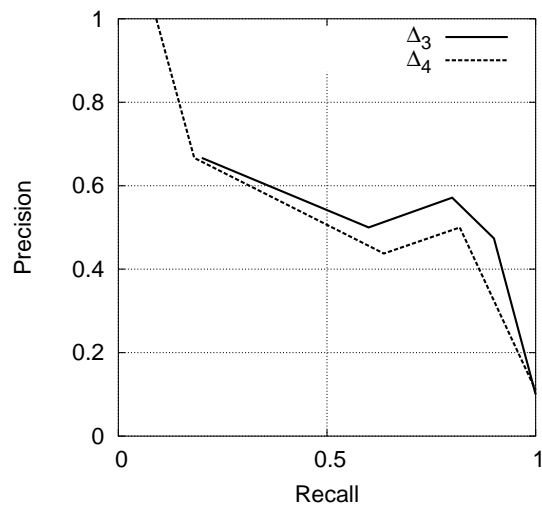


Abbildung 3.7: Recall-Precision-Graphen (mit linearer Interpolation) zu Δ_3 und Δ_4

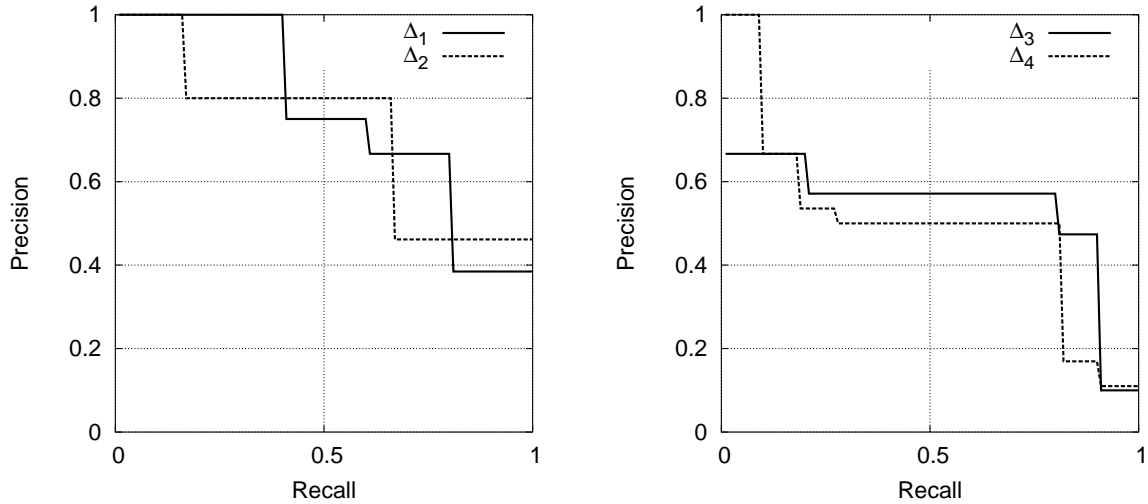


Abbildung 3.8: $PRECALL_{ceiling}$ bei linearer (links) und schwacher Ordnung (rechts)

Dokumenten sind das die Punkte $1/n \dots n/n$. Für die übrigen Recall-Werte wird dann jeweils die Precision des nächstgrößeren einfachen Recall-Punktes angenommen.

Sei l_f der letzte berücksichtigte Rang. Die $PRECALL_{ceiling}$ -Interpolation ist dann definiert durch:

- j := Anzahl irrelevanter Dokumente in den Rängen $\{1, \dots, l_{f-1}\}$
- s := Anzahl relevanter Dokumente, die aus Rang l_f gezogen werden
- r := Anzahl relevanter Dokumente in Rang l_f
- i := Anzahl irrelevanter Dokumente in Rang l_f

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

Untersuchen wir z. B. $\Delta_1 = (+ - - | + + + - - - - -)$ beim Recall-Level 0.25, ■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 einer Distribution $\Delta' = (+ - - | + + + - - - - -)$ nichts ändert, wohl aber bei einem Ergebnis $\Delta'' = (+ - - | + + + - - -)$.

Abbildung 3.8 zeigt die aus den Beispieldistributionen resultierenden Kurven für das $PRECALL_{ceiling}$ -Maß.

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_s)

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.

$$esl_s = \sum_{v=0}^i P'(v) \cdot v$$

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]):

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

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 $NR + esl_{NR}$ in die normale Formel zur Berechnung der Precision ein. NR bezeichne dabei die Anzahl gewünschter relevanter Dokumente; die erwartete Suchlänge ergibt sich dafür zu $esl_{NR} = j + esl_s$:

$$PRR := \frac{NR}{NR + esl_{NR}} = \frac{NR}{NR + j + s \cdot i / (r+1)}$$

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]):

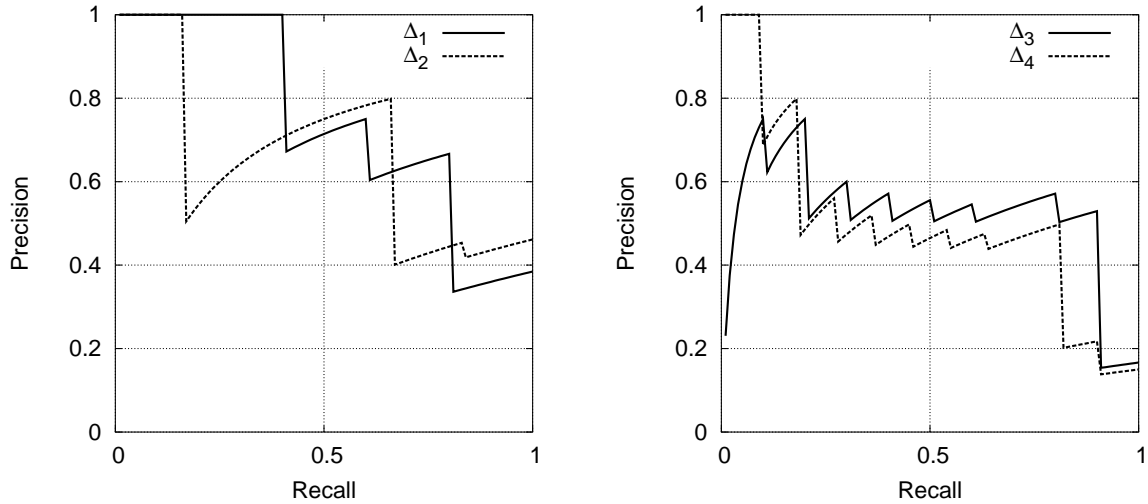


Abbildung 3.9: Probability of Relevance (NR) für lineare Ordnungen (links) und schwache Rangordnungen (rechts).

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

Für das Retrievalergebnis $\Delta = (+ - - | + + + - - - - - - -)$ erhalten wir z. B. für den Recall-Level 0.25, also $NR=1$, folgendes Ergebnis:

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

Abbildung 3.9 zeigt die mittels PRR errechneten Kurven für die Beispieldistributionen Δ_1 bis Δ_4 .

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 die Menge der möglichen Anordnungen, so berechnet sich die erwartete Precision zu:

$$EP_{NR} := \sum_{v \in V} P(v) \cdot p(v),$$

wobei $P(v) = \frac{1}{|V|} = \frac{r! \cdot s!}{(r+s)!}$ die Wahrscheinlichkeit einer Anordnung darstellt und $p(v)$ die Precision am NR -ten Dokument innerhalb der Anordnung v .

Als Beispiel diene wieder die Distribution $\Delta = (+ - - | + + + - - - - - - -)$. Die erwartete Precision für das erste relevante Dokumente EP_1 berechnet sich somit zu

$$EP_1 = \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{3} = \frac{11}{18} \approx 0,611.$$

Die folgende äquivalente Interpretation läßt sich einfacher berechnen, da zur Berechnung die Permutationen, für die ja jeweils die Precision-Werte zu bestimmen wären, nicht aufgezählt werden müssen. Zunächst wird wieder der Rang l_f bestimmt, aus dem das NR -te relevante Dokument stammt.

- t_r := Anzahl relevanter Dokumente in den Rängen $\{1, \dots, l_{f-1}\}$
- s := Anzahl relevanter Dokumente, die aus Rang l_f gezogen werden
($NR = t_r + s$)
- r := Anzahl relevanter Dokumente in Rang l_f
- i := Anzahl irrelevanter Dokumente in Rang l_f

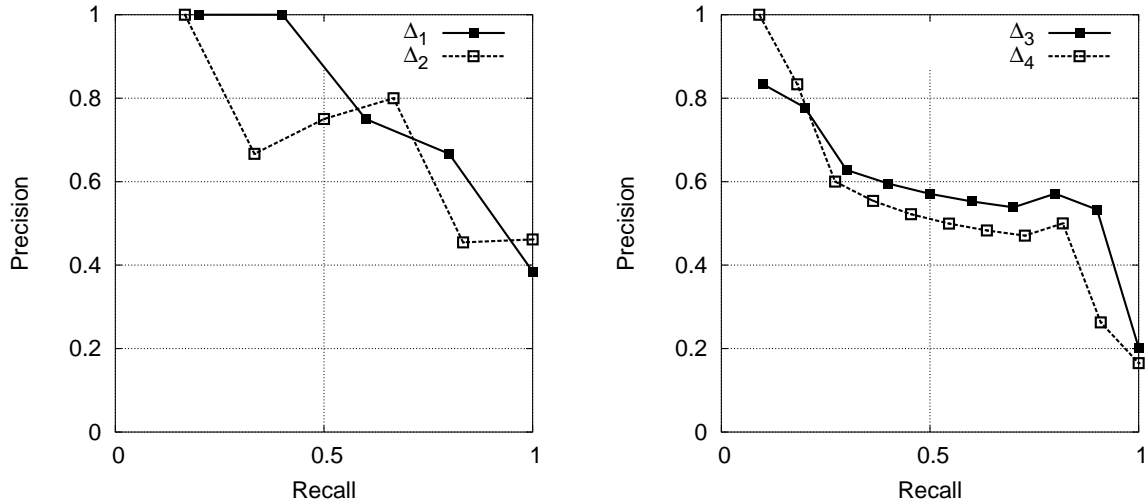


Abbildung 3.10: Expected Precision (NR) für lineare Ordnungen (links) und schwache Rangordnungen (rechts).

$$EP_{NR} := \sum_{k=s}^{r+i} P(v_{k,s}) \cdot p(v_{k,s})$$

Dabei bezeichnet $P(v_{k,s})$ die Wahrscheinlichkeit, dass sich in den ersten $k - 1$ aus Rang l_f gezogenen Dokumenten genau $s - 1$ relevante befinden und das als k -tes Dokument wiederum ein relevantes Dokument gezogen wird. $p(v_{k,s})$ bezeichnet die Precision an der Stelle:

$$EP_{NR} := \sum_{k=s}^{s+i} \frac{\binom{r}{s-1} \binom{i}{k-s}}{\binom{r+i}{k-1}} \cdot \frac{r-s+1}{r+i-k+1} \cdot \frac{s+t_r}{k+t_r+j}$$

Abbildung 3.10 zeigt die mittels Expected Precision errechneten Kurven für die Beispieldistributionen Δ_1 bis Δ_4 .

3.7.2 Abbruchkriterium: Anzahl der Dokumente

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

$$\begin{aligned} t_r &:= \text{Anzahl relevante Dokumente in den Rängen } \{1, \dots, l_f - 1\} \\ k &:= \text{Anzahl Dokumente, die aus Rang } l_f \text{ gezogen werden} \end{aligned}$$

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

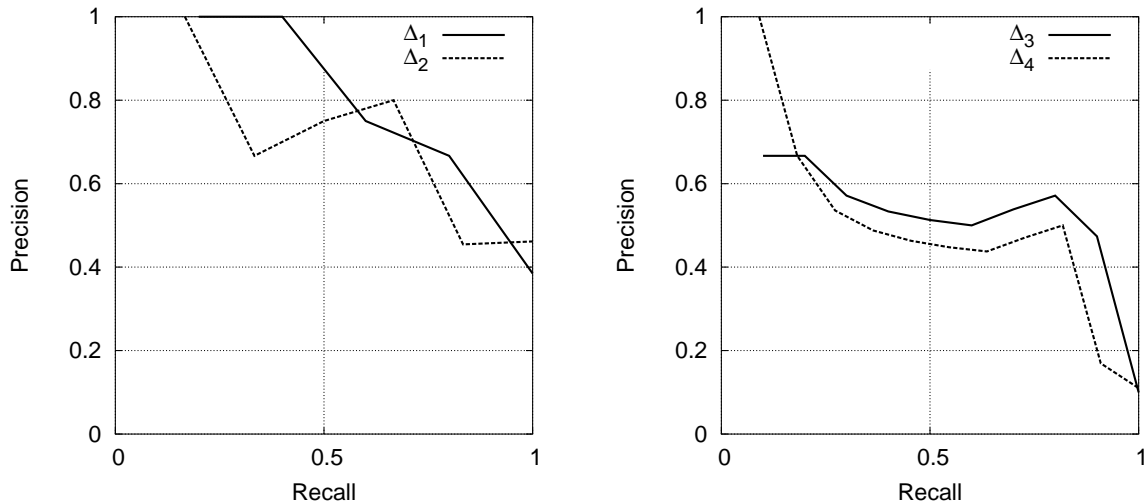
Definieren wir analog den *erwarteten Recall* ER

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

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

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

Abbildung 3.11 zeigt die für den Benutzerstandpunkt *NR* errechneten Kurven für die Beispieldistributionen Δ_1 bis Δ_4 .



Abbildungung 3.11: Benutzerstandpunkt *ND*. Für lineare Ordnungen (links) unterscheiden sich die Graphen nicht von den jeweiligen Graphen mit Benutzerstandpunkt *NR*, wohl aber für schwache Ordnungen (rechts).

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:

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 Evaluierungsinitiativen: TREC, CLEF, NTCIR und INEX

Basierend auf den Erfahrungen aus Evaluierungen in der Cranfield-Tradition [Cleverdon 91] entstand 1992 die TREC¹-Initiative, die bis heute einen Defacto-Standard für die Evaluierung von IR-Methoden darstellt.

¹<http://trec.nist.gov/>

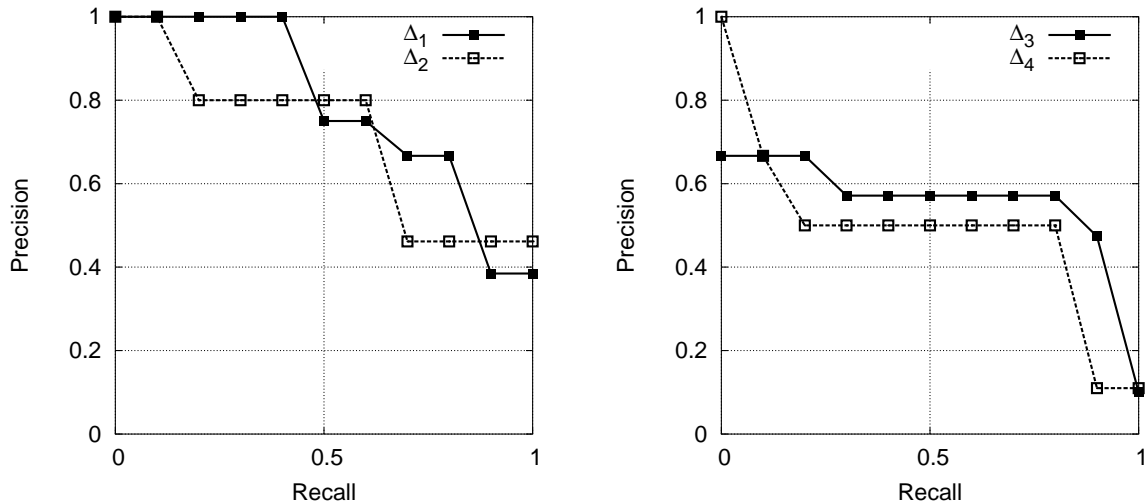


Abbildung 3.12: TREC-Evaluierungsmaße für lineare (links) und schwachgeordnete Rangordnungen (rechts).

Mittlerweile gibt es weitere Evaluierungsinitiativen, die jeweils darauf abzielen, für einen spezifischen Bereich im Information Retrieval eine Standardumgebung für die Evaluierung zur Verfügung zu stellen. Die wichtigsten dieser neueren Initiativen stellen CLEF², NTCIR³ und INEX⁴ dar.

Allen Initiativen gemein ist, dass sie zum einen Dokumentensammlungen in einem Umfang bereitstellen, der praktischen Anwendungen entspricht. Zum anderen werden standardisierte Retrievalaufgaben (Anfragen, sogenannte *Topics*) sowie entsprechende Relevanzbeurteilungen für die angebotenen Dokumentensammlungen ermittelt. Dabei werden die Relevanzbeurteilungen jeweils durch das in Abschnitt 3.5.2 beschriebene Pooling-Verfahren ermittelt. Es wird somit eine Vergleichbarkeit von IR-Verfahren hergestellt.

Bevor die einzelnen Initiativen mit ihren jeweiligen Teilaufgaben vorgestellt werden, soll im folgenden Abschnitt auf die verwendeten Retrievalmaße eingegangen werden.

3.9.1 Evaluierungsmaße

Evaluierung innerhalb der drei vorgenannten Initiativen basiert jeweils auf dem gleichen Retrievalmaß. Dabei wird für jeden Rang die Precision bestimmt, es erfolgt eine Interpolation nach dem Salton-Verfahren. Nun wird die Precision für die elf Recall-Punkte $\{0, 0.1, 0.2, \dots, 1\}$ wiederum unter Zuhilfenahme der Salton-Interpolation bestimmt. Die resultierenden Punkte werden wieder durch Geradenstücke miteinander verbunden. Bei mehreren Anfragen wird die Precision über die 11 Recall-Punkte gemittelt. Abbildung 3.12 zeigt die resultierenden Kurven für die Beispieldistributionen aus Abschnitt 3.6.

Bemerkenswert hierbei ist, dass eine Interpolation bis zum Recall-Punkt 0 erfolgt, für den ja die Precision eigentlich undefiniert sein muss. In der Implementierung dieses Maßes, wie sie in den einzelnen Initiativen Verwendung findet, werden schwachgeordnete Dokumente innerhalb einzelner Ränge ignoriert. Stattdessen wird die schwache Ordnung durch zufällige Anordnung der Dokumente eines Ranges in eine lineare Ordnung überführt.

Basierend auf dieser linearen Ordnung werden dann üblicherweise folgende globale Maße für ein System berechnet:

Benutzerorientierte Maße: Hier werden die Precision-Werte nach bestimmten Anzahlen gefundener Dokumente berechnet, also z.B. $\text{Prec}@5$, $\text{Prec}@10$, $\text{Prec}@30$, $\text{Prec}@100$. Aus den fragespezifischen Werten wird dann das jeweilige arithmetische Mittel (Makro-Mittelwert) über alle Fragen gebildet.

²<http://www.clef-campaign.org/>

³<http://research.nii.ac.jp/ntcir/>

⁴<http://www.is.informatik.uni-duisburg.de/projects/inex/index.html.en>

Bei bestimmten Aufgabenstellungen (z.B. Home Page Search) ist der Benutzer nur an einem relevanten Dokument interessiert. Hierfür wird der *Mean reciprocal rank* wie folgt berechnet: man bestimmte den Rang k des ersten relevanten Dokumentes zu jeder anfrage und bildet den Kehrwert $1/k$. Diese Kehrwerte werden dann über alle Fragen gemittelt (dies entspricht also dem harmonischen Mittel über die Ränge der ersten relevanten Dokumente).

Systemorientiertes Maß: Als globales systemorientiertes Maß wird der *Mean average precision (MAP)* verwendet. Hierbei wird der Mittelwert der Precision nach jedem relevanten Dokument einer Rangliste bestimmt und anschließend arithmetisch über alle Fragen gemittelt.

3.9.2 TREC: Text REtrieval Conference

Seit 1992 wird die jährliche *Text REtrieval Conference* [Voorhees & Harman 98] durchgeführt. Für die erste TREC-Konferenz wurden den Teilnehmern zwei Teilaufgaben (sogenannte *Tracks*) angeboten. Zum einen war das der *Adhoc-Track*, bei dem die Effektivität der Systeme in Bezug auf vorher nicht bekannte Anfragen gemessen wurde. Zum anderen gab es den *Routing-Track*. Hier wird die Effektivität von Systemen in Bezug auf Anfragen untersucht, zu denen bereits eine Menge relevanter Dokumente bekannt sind. Dabei können z. B. *Relevanz-Feedback-Methoden* für die Verbesserung der Retrieval-Ergebnisse auf einer Kollektion neuer Dokumente mit denselben Anfragen eingesetzt werden.

Mittlerweile wurde das Angebot an Tracks breit diversifiziert. So gibt es einen Track, der sich mit multilingualem Retrieval beschäftigt (*Cross-Language (CLIR) Track*). Weiterhin im Angebot sind z. B. ein Track für das Retrieval von Video-Daten, ein Web-Track (Retrieval von Web-Seiten) sowie der *Question-Answering Track*, bei dem es nicht mehr nur um das klassische Dokumentenretrieval geht, sondern um die möglichst exakte Extraktion des Wissens, welches eine vorgegebene Anfrage beantwortet.

3.9.3 CLEF: Cross-Language Evaluation Forum

Erstmals im Jahr 2000 fand das *Cross-Language Evaluation Forum (CLEF)* statt [Peters 01]. Hervorgegangen ist CLEF aus dem CLIR-Track von TREC: Der Schwerpunkt liegt auf der Evaluierung von Retrieval-Methoden für eine multilinguale Umgebung. Als Initiative auf Europäischer Ebene befasste sich CLEF zunächst nur mit den wichtigsten europäischen Sprachen. Ursprünglich waren dies Englisch, Französisch, Italienisch und Deutsch. Mittlerweile fand eine Ausweitung auf weitere Sprachen statt: Während Textkollektionen für die oben genannten Sprachen und Spanisch vorliegen, werden die Anfragen auch nach Niederländisch, Finnisch, Spanish, Swedish, Japanisch und Chinesisch übersetzt.

Grundsätzliches Ziel bei den CLEF-Konferenzen ist also die Evaluierung von Systemen, die relevante Dokumente auch in anderen Sprachen als die der Anfrage zurück liefern. Neben der Aufgabe für multilinguales Retrieval (Anfragen in einer Sprache, Dokumente in mehreren unterschiedlichen Sprachen) werden Aufgaben für bilinguales Retrieval angeboten. In Hinblick auf das monolinguale Retrieval im Rahmen der CLEF-Initiative werden spezielle Aspekte von Nicht-Englischen Sprachen untersucht.

3.9.4 NTCIR: NACSIS Test Collection Project

Ebenfalls mit Information Retrieval in einer multilingualen Umgebung befasst sich der NTCIR-Workshop, der zum ersten Mal in 1999 statt fand. Der Schwerpunkt liegt hierbei auf asiatischen Sprachen. Derzeit liegen Dokumentkollektionen in Japanisch, Chinesisch und Koreanisch vor. Ähnlich wie bei TREC gibt es hier neben der Aufgabe für multilinguales Retrieval weitere Aufgaben, z. B. Aufgaben zum Web-Retrieval, zum Patent-Retrieval sowie eine Question-Answering-Aufgabe.

3.9.5 INEX: Initiative for the Evaluation of XML Retrieval

Die jüngste der hier genannten Evaluierungsinitiativen ist die 2002 gestartete INEX-Initiative. Sie beschäftigt sich mit der Evaluierung von Retrievalmethoden für XML-Dokumente. Während bisherige Retrievalansätze Dokumente als atomare Einheiten auffassen, wird bei INEX die durch das XML-Format explizit gemachte hierarchische Dokumentstruktur berücksichtigt. Als Testkorpus werden hier Volltexte aus Informatik-Fachzeitschriften verwendet. Bislang werden zwei Arten von Aufgaben betrachtet:

- Bei 'content-only queries' ist eine natürlichsprachige Anfrage (i.w. eine Liste von Begriffen) gegeben, für die das IRS relevante Dokumentteile finden soll; diese sollen gerade so groß sein, dass sie die Anfrage beantworten, umfassendere Dokumentteile sollen dagegen vermieden werden.
- 'Content-and-structure queries' enthalten zusätzlich Bedingungen bezüglich der Dokumentstruktur, indem bestimmte Dokumentteile als Antwort verlangt werden (z.B. nur die Autoren oder die Titel) oder indem das Vorkommen von Werten in bestimmten Dokumentfeldern gefordert wird (z.B. Jahreszahl als Erscheinungsjahr, Autorname in den Zitationen).

Diese neue Art von Aufgaben verlangt auch nach neuen Evaluierungsmaßen, die sich allerdings noch in der Entwicklung befinden.

3.10 Evaluierung von interaktivem Retrieval

Bisher wurde in diesem Kapitel fast ausschließlich die Evaluierung von Batch-artigem Retrieval betrachtet - das trifft auch in überwiegendem Maße auf die oben beschriebenen Evaluierungsinitiativen zu. Dabei wird angenommen, dass der Benutzer eine Anfrage formuliert, und dann wird die Qualität der von den einzelnen Systemen produzierten Ergebnisse bestimmt. Dieser Ansatz hat allerdings eine Reihe von Schwächen:

- Es wird nur eine einzige Anfrage betrachtet, eine Reformulierung (wie sie bei interaktiven Systemen üblich ist) wird nicht berücksichtigt.
- Auch bei Relevance Feedback ist die einzig mögliche Interaktion die Relevanzbeurteilung einiger Dokumente, weitergehende Reaktionen des Benutzers (wie etwa Markierung relevanter/irrelevanter Passagen) sind nicht möglich.
- Heutige IR-Systeme bieten oft eine reichhaltige Funktionalität, wie z.B. Highlighting, Clustering, Browsing von Dokumenten oder Termlisten. Diese Funktionalität wird bei der Evaluierung nicht berücksichtigt.
- Ergebnisse aus dem TREC interactive track [Voorhees & Harman 00] zeigen, dass die in herkömmlichen Evaluierungen beobachteten Qualitätsunterschiede zwischen Verfahren beim interaktiven Retrieval verschwinden, da sie durch den Benutzer leicht kompensiert werden können [Turpin & Hersh 01].

Somit ergibt sich der Schluss, dass Ergebnisse aus Batch-Evaluierungen nur sehr beschränkte Aussagekraft auf die viel realistischere Situation des interaktiven Retrieval haben. Daraus ergibt sich die Notwendigkeit für die Evaluierung von interaktivem Retrieval.

Natürlich ist diese Art der Evaluierung sehr viel aufwendiger, da man dafür jeweils eine Reihe von Versuchspersonen benötigt. Auch die Auswertung von interaktivem Retrieval ist mit sehr viel Aufwand verbunden und wesentlich komplexer als im Batch-Fall. Mittlerweile gibt es ein reiches Instrumentarium für diese Art der Evaluierung:

- Bei *'think aloud'-Protokollen* soll die Versuchsperson laut denken, um damit mehr Einblick in die bei der Suche ablaufenden kognitiven Prozesse zu bekommen.
- *Beobachtungsdaten* (z.B. Log-Analyse) sind relativ einfach zu erheben, besitzen aber nur eine beschränkte Aussagekraft.
- Durch *Interviews* nach dem Versuch (und evtl. auch schon vorher) lässt sich der subjektive Eindruck der Versuchspersonen erheben und Hinweise auf die subjektiv empfundenen Stärken und Schwächen des Systems sammeln.
- *Fragebögen* können alternativ oder ergänzend zu Interviews eingesetzt werden. Sie erfordern weniger Aufwand für die Versuchsleitung, sind leichter auszuwerten und ermöglichen eine quantitative Beurteilung nach verschiedenen Kriterien.
- *Fehleranalysen* dienen dazu, bei der fehlgeschlagenen Bearbeitung von Aufgaben mit dem System Rückschlüsse auf die Ursachen zu ziehen.
- *Zeitbedarf zur Problembearbeitung* ist eine relativ einfach zu erhebende Messgröße: Für eine vorgegebene Menge von Aufgaben misst man jeweils die Zeit, die die Versuchspersonen zu deren Bearbeitung benötigen.
- Die *Kosten-Nutzen-Analyse* versucht, über die reine Retrievalqualität hinaus sowohl den Aufwand des Benutzers als auch den konkreten Nutzen zu quantifizieren.

Mittlerweile wird die Notwendigkeit der Evaluierung von interaktivem Retrieval allgemein anerkannt, allerdings wird der Aufwand zur Durchführung vielfach noch gescheut.

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. Dies liegt daran, 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 in diesem Kapitel zwei Arten von Lösungsansätzen für dieses Problem vorstellen:

- **semantischer Ansatz:**
Durch die Zuordnung von Deskriptionen zu Texten wird versucht, eine Repräsentation zu erstellen, die weitgehend unabhängig von der konkreten Formulierung im Text ist. Syntax und Semantik solcher Deskriptionen sind in Form sogenannter Dokumentationssprachen festgelegt.
- **Freitextsuche:**
Hierbei wird keine zusätzliche Repräsentation erstellt, sondern es werden nur bestimmte Funktionen zur Verbesserung der Suche im Text der Dokumente angeboten.

4.2 Freitextsuche

Bei der Freitextsuche kann man zwischen den beiden folgenden Ansätzen unterscheiden:

- **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 Zeichenebene.
- **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).

Bei beiden Ansätzen 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. **Stoppwortbestimmung:** Nicht-bedeutungstragende Wörter wie Artikel, Füllwörter oder Konjunktionen werden meist aus Aufwandsgründen von der weiteren Verarbeitung ausgeschlossen. Nur für syntaktische Verfahren müssen die Stoppwörter berücksichtigt werden, um ein korrektes Parsing zu ermöglichen. 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 (zusammengesetzte Wörter)
Donaudampfschiffahrtsgesellschaftskapitän Bundeskanzlerwahl
- Nominalphrasen (aus mehreren Nomen zusammengesetzte Begriffe) *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 “information 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 computerlinguistische 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
 / ‚oder‘
B Leerzeichen
 → ‚zu‘

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

- 1) **IES** → **Y**
- 2) **ES** → **B** wenn *O / CH / SH / SS / ZZ / X vorangehen
- 3) **S** → **B** wenn * / E / %Y / %O / OA / EA vorangehen
- 4) **S'** → **B**
IES' → **Y**
ES' → **B**
- 5) **'S** → **B**
' → **B**
- 6) **ING** → **B** wenn ** / % / X vorausgehen
ING → **E** wenn %* vorausgehen
- 7) **IED** → **Y**
- 8) **ED** → **B** 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** → **B**, 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** → **B**, 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'	→	ß
	IES'	→	Y
	ES'	→	ß

Beispiele zu 4:

MOTHERS'	→	MOTHER
LADIES'	→	LADY
FLAMINGOES	→	FLAMINGO

Regel 5	'S	→	ß
	'	→	ß

Beispiele zu 5:

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

Regel 6	ING	→	ß , 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	→	ß , 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 (Nominalphrasen) eingesetzt. Hierzu sind zwei Probleme zu lösen:

1. Wortklassenbestimmung: Zuordnung der syntaktischen Kategorie zu einzelnen Wörtern.
2. Parsing: Erkennen der syntaktischen Struktur. Für das Problem der Erkennung von Komposita muss keine vollständige syntaktische Analyse vorgenommen werden; es genügt, ein *partielles Parsing* zur Extraktion der relevanten Teilstrukturen.

Nachfolgend beschreiben wir diese beiden Probleme etwas detaillierter.

AT	article
BEZ	“is”
CONJ	conjunction
IN	preposition
JJ	adjective
JJR	comparative adjective
MD	modal (can, have, may, shall...)
NN	singular or mass noun
NNP	singular proper noun
NNS	plural noun
PERIOD	.:?!
PN	personal pronoun
RB	adverb
RBR	comparative adverb
TO	“to”
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle, gerund
VBN	verb, past participle
VBP	verb, non 3rd singular present
VBZ	verb, 3rd singular present
WDT	<i>wh</i> -determiner (what, which)

Tabelle 4.1: Häufig verwendete Wortklassen (für Englisch)

4.2.2.3.1 Wortklassenbestimmung Für die Definition von Wortklassen gibt es keinen Standard. Tabelle 4.1 zeigt jedoch eine häufig verwendete Schema.

Um die Wortklassen in einem Text zu bestimmen, kann auf dieselben Datenquellen zurückgegriffen werden, die auch bei der morphologischen Analyse verwendet werden:

- Vollformen-Wörterbücher enthalten alle Flexionsformen von Wörtern; üblicherweise enthält der Eintrag zu einer Vollform auch die zugehörigen Wortklassen.
- graphematische Verfahren versuchen, aus der Wortendung und evtl. Präfixen auf die Wortklasse zu schließen. Wegen des grundsätzlichen Problems der Unvollständigkeit von Wörterbüchern sollten graphematische Verfahren in jedem Fall eingesetzt werden, um auch unbekannte Wörter klassifizieren zu können.

Ein einfaches Beispiel für ein graphematisches Verfahren ist die in Tabelle 4.2 dargestellte Zuordnung von Wortklassen anhand von Kuhlens Algorithmus zur Grundformreduktion. Leider liefern die meisten Regeln keine eindeutige Wortklassenzuordnung.

Ein kombiniertes lexikalisch-graphematisches Verfahren zur Wortklassenbestimmung wurde in [Mikheev 98] vorgeschlagen. Dabei wird für nicht im Vollformen-Wörterbuch enthaltene Wörter versucht, diese mithilfe von graphematischen Verfahren auf andere Wörterbucheinträge zurückzuführen. Dadurch lassen sich auch mit einem relativ kleinen Wörterbuch relativ gute Ergebnisse erzielen. Tabellen 4.3 und 4.4 zeigen einige (häufig zutreffende) Regeln dieses Verfahren. Dabei steht WB-Klasse für die im Wörterbuch eingetragene Wortklasse des reduzierten Wortes, und die letzte Spalte enthält die jeweilige Wortklasse

Nr.	Regel	Klasse
1	IES → Y	NNS/VBP
2	ES → B	NNS/VBP
3	S → B	NNS/VBP
4	S' → B	NNS
	IES' → Y	
	ES' → B	
5	' S → B	NN
	' → B	
6	ING → B	VBG
	ING → E	
7	IED → Y	VBD/VBN/JJ
8	ED → B	VBD/VBN/JJ
	ED → E	

Tabelle 4.2: Wortklassenzuordnung basierend auf dem Kuhlen-Algorithmus

der Vollform; sind hier mehrere Klassen eingetragen, so korrespondieren diese jeweils zur entsprechenden Position in der mittleren Spalte. Die Regeln sind nach fallender Genauigkeit geordnet, daher taucht die Endung 's' z.B. mehrfach auf

Präfix	WB-Klassen	Wortklassen
re	JJ NN VBG	JJ NN VBG
ex	NN	NN
self-	NN	NN
inter	JJ	JJ
non	JJ	JJ
un	RB	RB
dis	JJ	JJ
anti-	NN	JJ
de	JJ VBD VBN	JJ VBD VBN
in	RB	RB

Tabelle 4.3: Präfix-Regeln

Postfix	WB-Klassen	Wortklassen
ment	NN VB VBP	NN
ing	NN VB VBP	JJ NN VBG
ed	NN VB VBP	JJ VBD VBN
s	NN VB VBP	NNS VBZ
ly	JJ NN RB	RB
ness	JJ	NN
ship	NN	NN
able	NN VB VBP	JJ
s	NN	NNS

Tabelle 4.4: Postfix-Regeln

In wenig flektierten Sprachen haben aber sowohl lexikalische als auch graphematische Verfahren mit einem grundsätzlichen Problem zu kämpfen: Vollformen können zu mehreren Wortklassen gehören, z.B.:

*The boys **play** football* vs.

*She saw the new **play**.*

Dieses Problem lässt sich nur durch die zusätzliche Berücksichtigung des Kontextes lösen, etwa in unserem Beispiel:

AT NNS VBP/NN NN → VBP

PN VBD AT JJ NN/VBP → NN

Üblicherweise betrachtet man Folgen von zwei oder drei Wörtern (Bigramme, Trigramme) als Kontextinformation.

Allerdings lässt sich auch dadurch keine befriedigende Lösung erreichen. [Greene & Rubin 71] zeigten, dass selbst bei einem vollständigen Wörterbuch die Wortklassenzuordnung mit einem deterministischem Tagger nur 77 % korrekte Zuordnungen liefert.

Durch den Übergang zu einem statistischen Ansatz lassen sich jedoch wesentlich bessere Ergebnisse erzielen. Dabei nutzt man die unterschiedliche Häufigkeit des Vorkommens in den verschiedenen Wortklassen aus (die meisten Wörter kommen in einer bevorzugten Wortklasse vor). Z.B. sind folgende Vorkommen eher selten:

to **flour** a pan

to **web** the final report

Ein einfacher Ansatz besteht daher darin, seltene Verwendungen zu ignorieren. So zeigten [Charniak et al. 93], dass sich durch dieses Vorgehen 90 % korrekte Zuordnungen erreichen lassen. Weitere Verbesserungen sind durch statistische Ansätze zur Berücksichtigung der syntaktischen Struktur (z.B. Markov-Modelle) möglich, wodurch sich etwa 95...97 % korrekte Zuordnungen erzielen lassen.

S	→	NP VP
NP	→	AT? JJ* NNS+
	→	AT? JJ* NN+
	→	NP PP
VP	→	VB PP
	→	VBZ
	→	VBZ NP
PP	→	IN NP

Tabelle 4.5: Einfache Beispielgrammatik

4.2.2.3.2 Parsing Basierend auf den zugeordneten Wortklassen kann man anschließend die syntaktische Struktur eines Textes bestimmen. Tabelle 4.5 zeigt eine einfache Grammatik (? steht für 0/1 Vorkommen, * für beliebig viele und + für mindestens einmaliges Vorkommen). Mit dieser Grammatik lassen sich die nachstehenden Beispielsätze analysieren:

- *The analysis of 25 indexing algorithms shows consistent retrieval performance.*
AT NN IN JJ NN NNS VBZ JJ NN NN
- *A good indexing technique for Web retrieval is manual classification.*
AT JJ NN NN IN NN NN VBZ JJ NN

4.2.2.3.2.1 Partielles Parsing Um Nominalphrasen beim Freitextretrieval zu erkennen, reicht in der Regel partielles Parsing aus. Dazu definiert man die relevanten syntaktischen Teilstrukturen. Lassen wir z.B. die Unterscheidung NN/NNP/NNS fallen, so könnte man folgende einfache Muster für Nominalphrasen definieren:

phrase → NN NN+
→ NN+ IN JJ* NN+

Damit kann man folgende Phrasen erkennen:

indexing algorithms

retrieval performance

retrieval of Web documents

retrieval of new documents

4.2.2.3.2.2 Head-Modifier-Strukturen Ein Matching von Nominalphrasen auf der Ebene der syntaktischen Strukturen führt in der Regel zu unbefriedigenden Ergebnissen. Ein besserer Ansatz ist die Transformation der Nominalphrasen in sogenannte Head-Modifier-Strukturen. Für eine zweigliedrige

Nominalphrasen bezeichnet dabei *head* das Nomen, das die wesentliche Bedeutung des Kompositums ausdrückt, z.B. *information retrieval*, *indexing algorithm*. Der *modifier* dagegen spezialisiert oder modifiziert die Bedeutung des heads.

Bei mehr als zweigliedrigen Ausdrücken ergeben sich geschachtelte Strukturen, die man in Listen- oder Baum-Form darstellen kann (siehe auch Abbildung 4.1). Dabei steht jeweils der Modifier links und der Head rechts:

$((\text{multimedia}, \text{document}), \text{retrieval}), \text{system}$)

the domain of possible categories of linguistic expressions

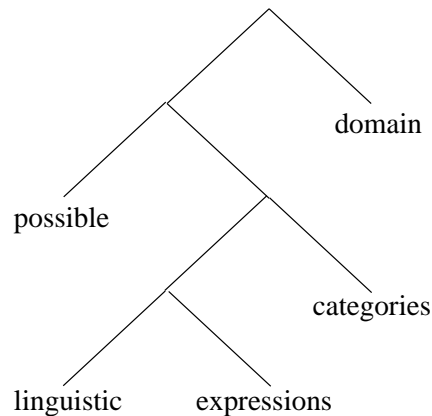


Abbildung 4.1: Beispiel für geschachtelte Head-Modifier-Struktur im Englischen

4.2.2.3.2.3 Matching-Prozess Der Vergleich zwischen einem Kompositum aus der Anfrage und einem im Dokumenttext gefundenen läuft nun wie folgt ab:

1. Nominalphrasen Komposita in Head-Modifier-Struktur überführen.

Die Transformationsregeln basieren dabei primär auf der syntaktischen Struktur

2. Test, ob das Anfragewort in der Nominalphrase aus dem Dokument enthalten ist.

Dabei müssen Head- bzw. Modifier-Rolle bzgl. der gemeinsamen Wurzel übereinstimmen. Ein einzelnes Nomen wird dabei als Head aufgefasst.

Zum Beispiel ist der Dokumentterme $((\text{semistructured}, \text{data}), \text{retrieval}) \text{ system}$ ein Treffer bzgl. der Anfrageterme $(\text{retrieval}, \text{system})$, $(\text{semistructured}, \text{data})$ und $(\text{data}, \text{retrieval})$, aber nicht für $(\text{retrieval}, \text{data})$.

4.3 Dokumentations Sprachen

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 Dokumentations Sprachen, nämlich Klassifikationen und Thesauri. Diese Ausführungen orientieren sich im wesentlichen an der Darstellung in [Burkart 90]. Anschließend wird als Beispiel für einen moderneren Ansatz die Sprache RDF 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 Randbe-

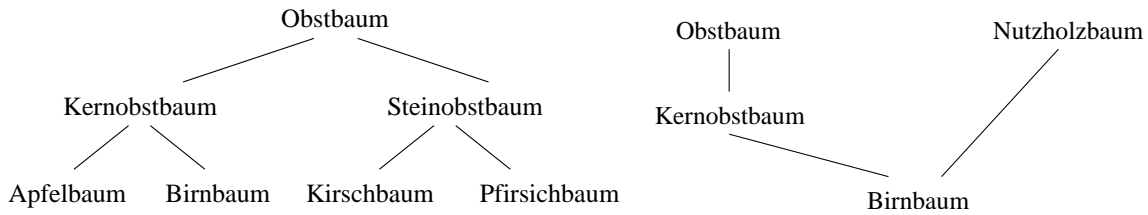


Abbildung 4.2: Monohierarchie (links) und Polyhierarchie (rechts)

dingung ergibt sich bereits eine prinzipielle Schwäche, da viele Dokumente ja gerade versuchen, Brücken zwischen verschiedenen 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.

Die bekanntesten Beispiele für Klassifikationen sind die den Web-Katalogen (wie z.B. Yahoo!) zugrundeliegenden Ordnungssysteme. Daneben gibt es sehr viele fach- oder anwendungsspezifische Klassifikationen, wie z.B.

- LCC** Library of Congress Classification
- DDC** Dewey Decimal Classification
- UDC** Universal Decimal Classification
- MSc** Mathematics Subject Classification
- CCS** ACM Computing Classification system

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 4.2 zeigt links 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.

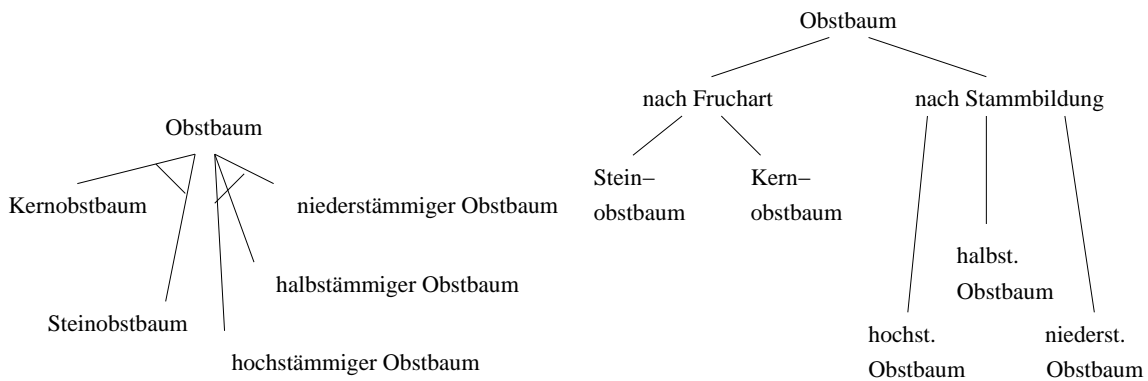


Abbildung 4.3: Polydimensionalität

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 Klassifikation, wie das Beispiel in Abb. 4.3 links illustriert, erlaubt die Darstellung dieses Sachverhaltes. Erlaubt das Klassifikationsschema keine Polydimensionalität, dann muss diese durch Einführung einer zusätzlichen Hierarchieebene aufgelöst werden (Abb. 4.3 rechts), 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 4.6: 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 4.6 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 Die Yahoo-Klassifikation

Abbildung 4.4 zeigt die Hauptklassen der Yahoo-Klassifikation, und Abbildung 4.5 die weitere Unterteilung der Hauptklasse „Computers & Internet“. Mit '@' markierte Klassen bezeichnen dabei Querverweise in der Klassenhierarchie. Das Ordnungssystem ist somit kein Baum, sondern ein gerichteter Graph. Typisch für Yahoo! ist ferner die variierende Tiefe des Ordnungssystems, die an manchen Stellen nur 3, an anderen bis zu 7 beträgt. Dabei können die zu klassifizierenden (Web-)Dokumente beliebigen Knoten zugeordnet werden. Somit enthält ein Knoten in der Regel die Verweise auf die zugehörigen Dokumente sowie die Liste der Unterklassen.

4.3.2.3 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.

Arts & Humanities Literature, Photography...	News & Media Full Coverage, Newspapers, TV...
Business & Economy B2B, Finance, Shopping, Jobs...	Recreation & Sports Sports, Travel, Autos, Outdoors...
Computers & Internet Internet, WWW, Software, Games...	Reference Libraries, Dictionaries, Quotations...
Education College and University, K-12...	Regional Countries, Regions, US States...
Entertainment Cool Links, Movies, Humor, Music...	Science Animals, Astronomy, Engineering...
Government Elections, Military, Law, Taxes...	Social Science Archaeology, Economics, Languages...
Health Medicine, Diseases, Drugs, Fitness...	Society & Culture People, Environment, Religion...

Abbildung 4.4: Yahoo!-Hauptklassen

Art@	Employment@
Bibliographies (6)	Ethics (18)
Communications and Networking (1146)	Games@
Computer Science@	Graphics (316)
Contests (26)	Hardware (2355)
Conventions and Conferences@	History (106)
Countries, Cultures, and Groups (38)	Humor@
Cyberculture@	Industry Information@
Data Formats (485)	Internet (6066)
Desktop Customization@	Magazines@
Desktop Publishing (53)	Mobile Computing (65)
Dictionaries (24)	Multimedia (690)
	Music@
	News and Media (205)
	...

Abbildung 4.5: Untergliederung der Hauptklasse Computers & Internet

4.3.2.3.1 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.

4.3.2.3.2 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*

4.3.2.3.3 Facettierende Elemente

Zur Facettierung in der DK dienen die Anhängenzahlen, die durch spezielle Zeichen eingeleitet werden. Es gibt einerseits allgemeine Anhängenzahlen, die überall in der DK verwendet werden dürfen, und andererseits spezielle Anhängenzahlen, die nur für bestimmte Klassen innerhalb der DK erlaubt sind. Beispiele für allgemeine Anhängenzahlen sind folgende (die jeweils einleitende Zeichenfolge ist vorangestellt):

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

4.3.2.3.4 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),
- ' Zusammenfassungszeichen zur Bildung neuer Sachverhalte aus der Kombination einzelner DK-Komponenten.

4.3.2.4 Computing Reviews Classification

Als weiteres 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.

4.3.2.4.1 General terms:

Die general terms der CR Klassifikation sind in Tabelle 4.7 aufgelistet.

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

Tabelle 4.7: General terms der CR Klassifikation

4.3.2.4.2 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 (natürlichen) 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 Bedeutungsunterschieden 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 Polysemkontrolle werden 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*

Information retrieval		Query processing	
<i>UF</i>	CD-ROM searching Data access Document retrieval Online literature searching Retrieval, information	<i>UF</i>	Data querying Database querying Query optimisation
<i>BT</i>	Information science	<i>BT</i>	Information retrieval
<i>NT</i>	Query formulation Query processing Relevance feedback	<i>RT</i>	Database management systems Database theory DATALOG Query languages
<i>RT</i>	Bibliographic systems Information analysis Information storage Query languages	Query formulation	
		<i>UF</i>	Search strategies
		<i>BT</i>	Information retrieval
		Relevance feedback	
		<i>BT</i>	Information retrieval

Abbildung 4.6: Auszug aus dem Beziehungsgefüge des INSPEC-Thesaurus'

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,

0.0058 Magnetband VB Magnetbandlaufwerk	Magnetismus (Forts.) BF Halleffekt BF Induktion OB Elektrodynamik UB Magnetfeld
0,0045 Magnetbandgerät BS Magnetbandlaufwerk NE7	BIK Geophysik BFK Erdmagnetismus BIK Optik BFK Faraday-Effekt
0. 0046 Magnetbandkassette NO NE83 BF Kassette BF MB-Kassette OB Datenträger VB Magnetbandkassettenlaufwerk	0.0070 Magnetkarte NO NE87 BF Telefonkärtchen OB Datenträger VB Kartensystem
0.0051 Magnetbandkassettengerät BS Magnetbandkassettenlaufwerk NE7	0.0073 Magnetkartensystem NO ECS OB Kartensystem
0.0050 Magnetbandkassettenlaufwerk NO NE7 BF Magnetbandkassettengerät BF MB-Kassettengerät OB Datenausgabegerät OB Dateneingabegerät OB Datenspeichertechnik VB Magnetbandkassette	0.0074 Magnetkartentelefon NO GK72 BF Makatel OB Kartentelefon
0.0044 Magnetbandlaufwerk NO NE7 BF Magnetbandgerät OB Bandgerät OB Datenausgabegerät OB Dateneingabegerät OB Datenspeichertechnik VB Magnetband	0 0077 Magnetplatte NO NE82 OB Datenspeicher OB Datenträger VB Magnetplattenlaufwerk BIK Datenspeicher BFK Plattenspeicher
0.0059 Magnetfeld NO WD2 OB Magnetismus	0.0081 Magnetplattengerät BS Magnetplattenlaufwerk NE7
0.0060 Magnetismus NO WD2 BF Barkhausen-Effek BF Ferromagnetismus	0.0079 Magnetplattenlaufwerk NO NE7 BF Magnetplattengerät OB Datenausgabegerät OB Dateneingabegerät OB Datenspeichertechnik VB Magnetplatte

Abbildung 4.7: Auszug aus einem Thesaurus

- Einführungs-/Streichungsdatum.

Abbildung 4.7 zeigt ein Beispiel für einen Ausschnitt 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 RDF (Resource Description Framework)

RDF ist eine vom W3C im Rahmen der 'Semantic Web'-Initiative geförderte Beschreibungssprache. Diese Initiative verfolgt die Vision einer weltweit verteilten Wissensbasis. Jede Web-Site kann dabei einen bestimmten Wissensausschnitt modellieren, und durch Kombination der für eine bestimmte Anwendung relevanten Ausschnitte erhält man mit geringem Aufwand eine sehr detaillierte Modellierung, mit deren Hilfe sich anspruchsvolle Aufgaben realisieren lassen.

Die grundlegende Idee ist dabei, eine im Vergleich zu Thesauri und Klassifikationen ausdrucksstärkere Beschreibungssprache zu entwickeln. Diese soll folgende Eigenschaften besitzen:

- Während bei der Deskribierung mittels Thesauri nur Konzepte zugeordnet werden können, erlaubt RDF auch die Zuordnung von Instanzen zu Konzepten (z.B. „Schröder“ als Instanz von „Bundeskanzler“)
- Neben den in Thesauri üblichen Beziehungen können zusätzlich beliebige Beziehungen eingeführt und zur Deskribierung verwendet werden.

- Anstelle der einfachen Zuordnung von Konzepten zu Dokumenten besteht die Beschreibungssprache aus Statements der Art Subjekt-Prädikat-Objekt.

4.3.4.1 RDF: Grundlegende Konzepte

RDF beinhaltet folgende Grundkonstrukte:

Resource Ressourcen bezeichnen beliebige Objekte im WWW, wie z.B. eine einzelne Web-Seite, oder auch ein umfangreiche Datenbasis. Ressourcen werden in der Regel durch einen Uniform Resource Identifier (URI) identifiziert.

Literal Literale sind spezielle Ausprägungen von Ressourcen, die eine Zeichenkette als Wert haben, aber keine explizite URI.

Property Eigenschaften bezeichnen eine gerichtete Beziehung zwischen zwei Ressourcen (Attribut, Relation, Rolle o.ä.).

Statement Aussagen sind ein Tripel bestehend aus Ressource, benannter Eigenschaften und einer weiteren Ressource, die den Wert für die Eigenschaft angibt. Somit entspricht die Syntax einer Aussage der einfacher Sätze der Form Subjekt – Prädikat – Objekt.

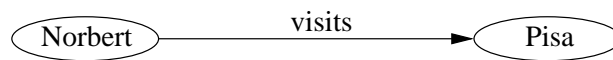


Abbildung 4.8: Eine einfache Aussage in RDF

Diese Konstrukte kann man graphisch darstellen, wobei Ressourcen als Ellipsen, Literale als Rechtecke und Eigenschaften als gerichtete Kanten dargestellt werden (siehe z.B. Abbildung 4.8).

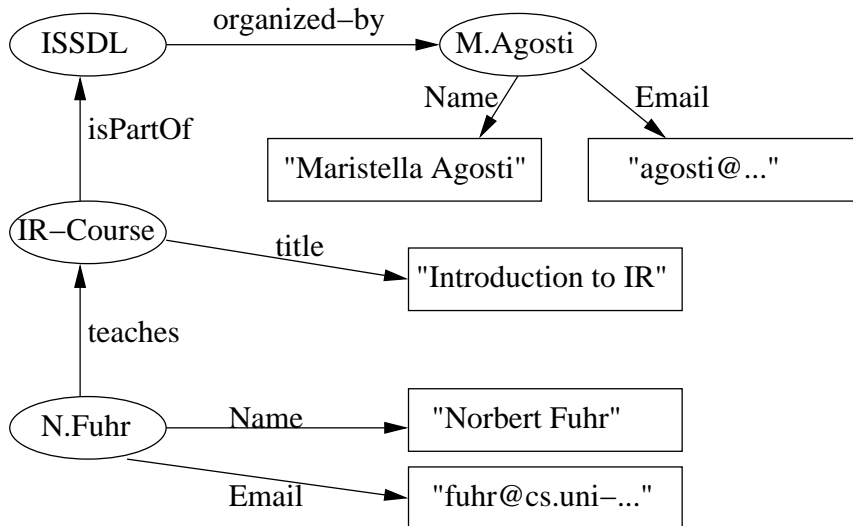


Abbildung 4.9: Ein RDF-Graph

Nehmen verschiedene Aussagen auf die selben Ressourcen Bezug, so erhält man einen RDF-Graphen wie z.B. in Abbildung 4.9. So intuitiv diese Darstellung auch sein mag, so ist doch offensichtlich, dass durch die uneingeschränkte Verwendung von Ressourcen und Eigenschaften das Ziel der Interoperabilität von RDF-Wissensbanken nicht erreicht werden kann. Man benötigt daher Mechanismen zur Deklaration der verwendeten Ressourcen und Eigenschaften, analog zu einem Datenbankschema.

4.3.4.2 RDF Schemas

RDF Schemas sind ähnlich zu denen objektorientierter Datenbanken. Der wesentliche Unterschied besteht darin, dass RDF Schemas nur selten Informationen über Datentypen enthalten, dagegen aber über stärkere Abstraktionsmechanismen verfügen. Sie stehen damit in einer langen KI-Tradition, die ihren Ursprung

in den semantischen Netzen (z.B. KL-ONE) hat und über terminologische bzw. Beschreibungs-Logiken schließlich zu RDF geführt hat.

RDF Schemas beschreiben die Beziehungen zwischen Typen von Ressourcen und/oder Eigenschaften. Dabei werden die Schemas selbst auch wieder mittels RDF beschrieben. Hierzu stehen folgende Grundkonstrukte zur Verfügung:

- Grundlegende Klassen:
 - rdfs:Resource bezeichnet die Klasse aller Ressourcen,
 - rdf:Property die Klasse aller Eigenschaften und
 - rdfs:Class die Klasse aller Klassen.
- Beziehungen zwischen verschiedenen Ressourcen, Eigenschaften oder Klassen können durch folgende Eigenschaften beschrieben werden:
 - rdf:type gibt den Typ eines Konstrukts an.
 - rdfs:subClassOf bezeichnet die Teilklassen-Beziehung.
 - rdfs:subPropertyOf erlaubt die Spezialisierung von Eigenschaften.
 - rdfs:seeAlso beschreibt Querverweise.
 - rdfs:isDefinedBy verweist von einer Ressource oder Eigenschaft auf die zugehörige Definition.

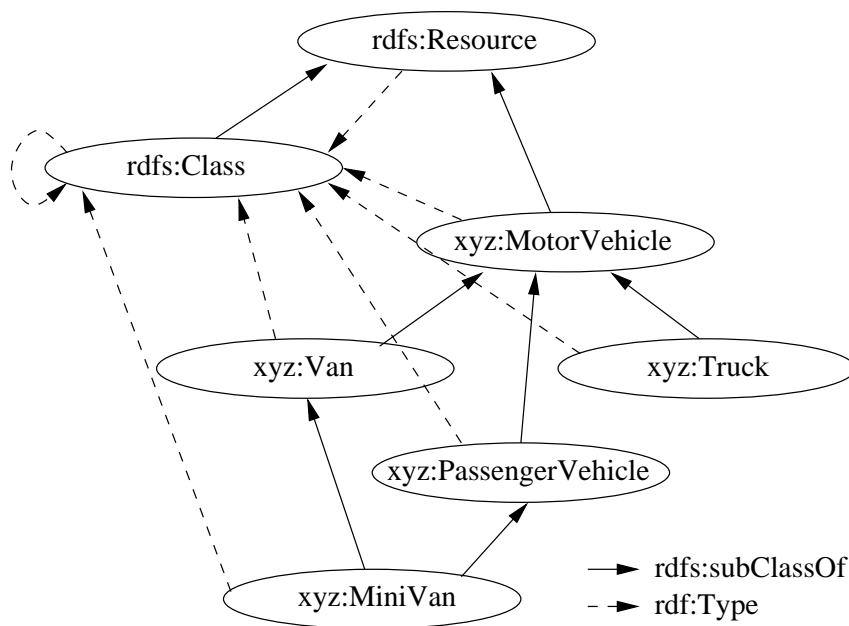


Abbildung 4.10: Eine Hierarchie von RDF-Ressourcen

Die Abbildungen 4.10 und 4.11 zeigen kleine Beispiele von RDF-Schemata.

4.3.5 Dokumentations Sprachen vs. Freitext

Beim Vergleich mit der Freitextsuche sind folgende Vor- und Nachteile von Dokumentations Sprachen 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 Dokumentations Sprache abbilden muss, ergibt sich eine größere Benutzerfreundlichkeit.
- Die Benutzung des Systems setzt die Kenntnis der Dokumentations Sprache voraus; für gelegentliche Benutzer ist diese Hürde zu hoch.

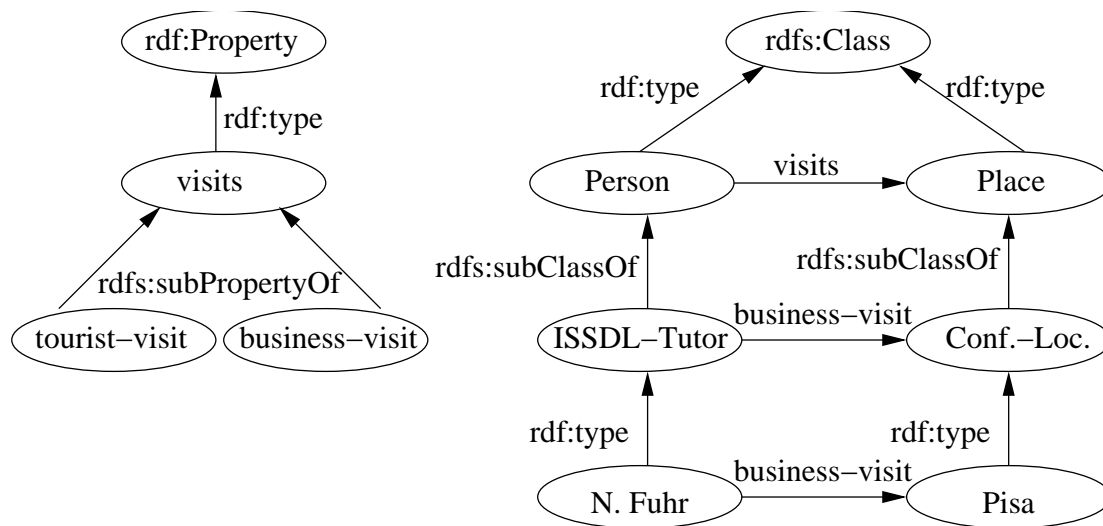


Abbildung 4.11: Hierarchien auf Ressourcen und Eigenschaften

- 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 Datenbanken 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 Datenbanken 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 Datenbanken mit mehreren GB Text angewendet und die Ergebnisse miteinander verglichen. Die auf den TREC-Konferenzen [Voorhees & Harman 00] präsentierten Ergebnisse zeigen, dass halb-formale Konzepte (wie z.B. geographische oder Datumsangaben) durch eine reine Freitextsuche nicht abzudecken sind, so dass zumindest für diesen 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 kaum erfolgversprechenden Ansätze in diesem Bereich. Das Hauptproblem ist das Fehlen entsprechender Wissensbasen, die nicht nur sehr umfangreich sein müssen, sondern auch das für die jeweilige Anwendung pragmatische Wissen bereitstellen sollten. Letzteres ist wohl der Hauptgrund, warum selbst so umfangreiche Wissensbasen wie CYC [Lenat et al. 90] bislang nicht erfolgreich im IR eingesetzt werden konnten.
- 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 $\vec{x} = (x_1, \dots, x_n)$ repräsentieren, wobei die Komponente x_i jeweils das Vorkommen des Terms $t_i \in T = \{t_1, \dots, t_n\}$ 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 $x_i \in \{0, 1, 2, \dots\}$.

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!

Kapitel 5

Nicht-probabilistische IR-Modelle

5.1 Notationen

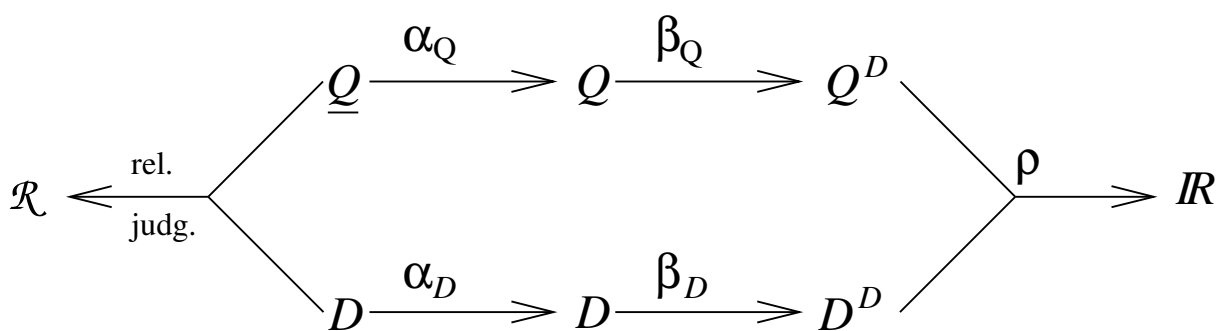


Abbildung 5.1: Konzeptionelles Modell für Textretrieval

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 2.2 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:

- $T = \{t_1, \dots, t_n\}$: Indexierungsvokabular
- q_k : Frage
- q_k : Frage-Repräsentation (formalisierte Anfrage)
- q_k^D : Frage-Beschreibung (Fragelogik)

- \underline{d}_m : Dokument
- \bar{d}_m : Dokument-Repräsentation (semantische Sicht)
- d_m^D : Dokument-Beschreibung (Objektattribute)
- $\vec{d}_m = \{d_{m_1}, \dots, d_{m_n}\}$: 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. Einklammerte 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.

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

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

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

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

1. $t_i \in T \Rightarrow \varrho(t_i, \vec{d}_m) = d_{m_i}$
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 Zerteilung 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.

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

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 Ordnung 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.
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

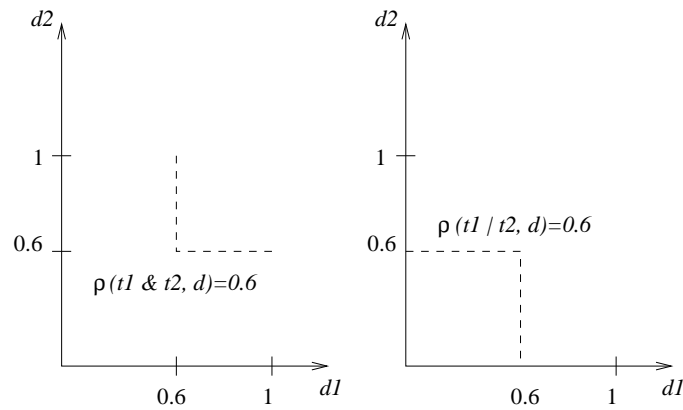


Abbildung 5.3: Punkte mit gleichem Retrievalgewicht beim Fuzzy-Retrieval

Modell werden hier bei den Dokumenten-Beschreibungen nun auch gewichtete Indexierungen zugelassen, d.h. $d_{m_i} \in [0, 1]$. Frage-Beschreibungen und Retrievalfunktion sind wie beim booleschen Retrieval definiert.

Durch die gewichtete Indexierung liefert die Retrievalfunktion jetzt Werte $\varrho(q_k^D, \vec{d}_m) \in [0, 1]$. 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 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. In [Lee et al. 93] werden die hierzu aus der Fuzzy-Theorie bekannten T-Normen sowie eigene Erweiterungsvorschläge evaluiert; dabei zeigt sich dass die hier vorgestellte Standarddefinition der Fuzzy-Operatoren relative schlecht abschneidet. Ein alternatives Modell ist unter dem Namen "Extended Boolean Retrieval" in [Salton et al. 83] beschrieben 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.)

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 Vektorraummodell

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):

$$d_m^D = \vec{d}_m \quad \text{mit} \quad d_{m_i} \in \mathbb{R} \quad \text{für} \quad i = 1, \dots, n \quad (5.2)$$

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

$$q_k^Q = \vec{q}_k \quad \text{mit} \quad q_{k_i} \in \mathbb{R} \quad \text{für} \quad i = 1, \dots, n \quad (5.3)$$

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

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

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
\neg aging	-2		1		
Retrievalgewicht		5	2	6	4.5

5.5.1 Coordination Level Match

Eine vereinfachte Variante des Vektorraummodells ist der Coordination Level Match. Dabei sind sowohl für Frage- als auch für Dokumenttermgewichtung nur die binären Werte 0 und 1 zugelassen. Die *Dokument-Beschreibung* ist somit die gleiche wie beim Booleschen Retrieval:

$$d_m^D = \vec{d}_m \quad \text{mit} \quad d_{m_i} \in \{0, 1\} \quad \text{für} \quad i = 1, \dots, n.$$

Die *Frage-Beschreibung* ist ebenfalls ein binärer Vektor:

$$q_k^Q = \vec{q}_k \quad \text{mit} \quad q_{k_i} \in \{0, 1\} \quad \text{für} \quad i = 1, \dots, n.$$

Als *Retrievalfunktion* verwendet man meist das Skalarprodukt; dadurch zählt die Retrievalfunktion die Anzahl der Frageterme, die im jeweiligen Dokument vorkommen:

$$\varrho(\vec{q}_k, \vec{d}_m) = \vec{q}_k \cdot \vec{d}_m = |q_k^T \cap d_m^T|$$

5.5.2 Relevance Feedback

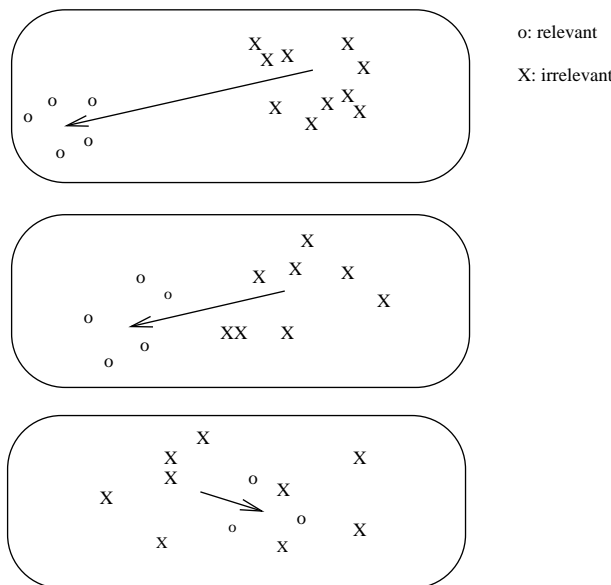


Abbildung 5.4: Trennung von relevanten und nichtrelevanten Dokumenten im VRM

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.

In [Rocchio 66] wird eine optimale Lösung für die Bestimmung eines Fragevektors aus Relevance-Feedback-Daten vorgestellt. Die Grundidee ist dabei die, einen Fragevektor \vec{q} zu bestimmen, der die Differenz der RSVs zwischen relevanten und irrelevanten Dokumenten maximiert. Sei D^R die Menge der relevanten Dokumente zu q und D^N die Menge der nichtrelevanten Dokumente zu q , dann lautet das Optimierungskriterium:

$$\sum_{(d_k, d_l) \in D^R \times D^N} \vec{q} \vec{d}_k - \vec{q} \vec{d}_l \stackrel{!}{=} \max \quad (5.5)$$

Zusätzlich muss man noch als Nebenbedingung den Betrag des Fragevektors beschränken:

$$\sum_{i=1}^n q_i^2 = c \quad (5.6)$$

Somit liegt ein Extremwertproblem mit Randbedingung vor, das man mit Hilfe eines Lagrange-Multiplikators lösen kann:

$$F = \lambda \left(\sum_{i=1}^n q_i^2 - c \right) + \sum_{(d_k, d_l) \in D^R \times D^N} \sum_{i=1}^n q_i d_{k_i} - q_i d_{l_i} \quad (5.7)$$

Zur Lösung muss man nun alle partiellen Ableitungen von F nach den Komponenten q_i des Fragevektors 0 setzen; zusätzlich muss auch die Nebenbedingung 5.6 gelten.

$$\begin{aligned} \frac{\partial F}{\partial q_i} &= 2\lambda q_i + \sum_{(d_k, d_l) \in D^R \times D^N} d_{k_i} - d_{l_i} \stackrel{!}{=} 0 \\ q_i &= -\frac{1}{2\lambda} \sum_{(d_k, d_l) \in D^R \times D^N} d_{k_i} - d_{l_i} \\ \vec{q} &= -\frac{1}{2\lambda} \sum_{(d_k, d_l) \in D^R \times D^N} \vec{d}_k - \vec{d}_l \\ &= -\frac{1}{2\lambda} |D^N| \sum_{d_k \in D^R} \vec{d}_k - |D^R| \sum_{d_l \in D^N} \vec{d}_l \\ &= -\frac{|D^N| |D^R|}{2\lambda} \frac{1}{|D^R|} \sum_{d_k \in D^R} \vec{d}_k - \frac{1}{|D^N|} \sum_{d_l \in D^N} \vec{d}_l \end{aligned}$$

Zur Vereinfachung wählen wir c (den Betrag des Fragevektors) so, dass $|D^N| |D^R| / 2\lambda = -1$. Damit ergibt sich der optimale Fragevektor zu

$$\vec{q} = \frac{1}{|D^R|} \sum_{d_k \in D^R} \vec{d}_k - \frac{1}{|D^N|} \sum_{d_l \in D^N} \vec{d}_l \quad (5.8)$$

Der optimale Fragevektor ist somit der Verbindungsvektor der beiden Zentroiden der relevanten bzw. irrelevanten Dokumente.

Abbildung 5.5.2 illustriert diese Lösung. Gleichzeitig wird deutlich, dass der optimale Fragevektor nicht immer die bestmögliche Lösung (bezogen auf die Retrievalqualität) darstellt. (Ein wesentlich besseres, allerdings auch aufwändigeres Verfahren ist die Support Vector Machine [Joachims 01].) Als heuristische Verbesserung, die sich in zahlreichen Experimenten bewährt hat, hat Rocchio vorgeschlagen, relevante

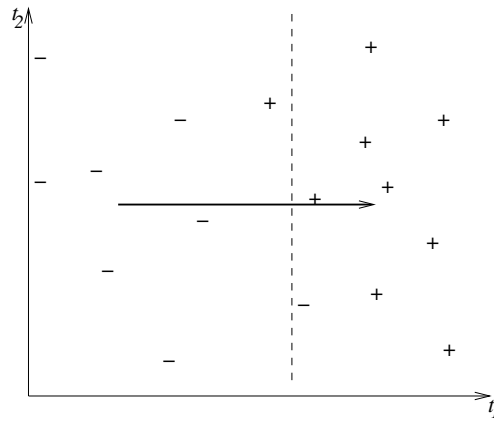


Abbildung 5.5: Optimaler Fragevektor als Verbindungsvektor der Zentroiden

und irrelevante Dokumente unterschiedlich stark zu gewichten, konkret: den Vektor zum Zentroiden der irrelevanten Dokumente weniger stark in die Lösung einfließen zu lassen. Abbildung 5.5.2 verdeutlicht diese Vorgehensweise für unser Beispiel. Intuitiv kann man sich diese Verbesserung dadurch erklären, dass in der Regel die relevanten Dokumente höhere Indexierungsgewichte als die irrelevanten aufweisen, so dass diese Modifikation den Fragevektor in die richtige Richtung „dreht“.

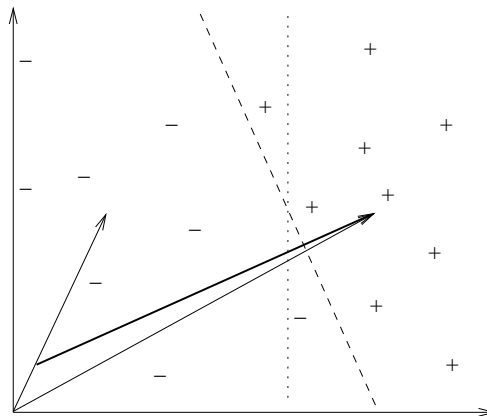


Abbildung 5.6: Unterschiedliche Gewichtung positiver und negativer Beispiele

Weitere Experimente haben gezeigt, dass man den neuen Fragevektor nie allein aus den Relevance-Feedback-Daten ohne Berücksichtigung des ursprünglichen Vektors bilden sollte; Es gibt ja noch weitere Dokumente, über die noch keine Relevanzinformation verfügbar ist, weil das System diese dem 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. Also 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}' 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}' wird in [Rocchio 66] folgende Kombination aus ursprünglichem Vektor \vec{q} und den Zentroiden der relevanten und der nichtrelevanten

Dokumente vorgeschlagen:

$$\vec{q}' = \vec{q} + \alpha \frac{1}{|D^R|} \sum_{d_j \in D^R} \vec{d}_j - \beta \frac{1}{|D^N|} \sum_{d_j \in D^N} \vec{d}_j \quad (5.9)$$

Dabei sind α und β nichtnegative Konstanten, die heuristisch festzulegen sind (z.B. $\alpha = 0.75$, $\beta = 0.25$).

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

Tabelle 5.2 zeigt experimentelle Ergebnisse, die durch Anwendung der Formel 5.9 gewonnen wurden (aus [Salton & Buckley 90]). Hier wurde Feedback-Information von den obersten 15 Dokumenten des Retrievalaufs 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 5.9 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 5.5 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).

5.5.3 Dokumentindexierung

Das VRM macht keine Aussagen darüber, wie die Dokumentenbeschreibung zu erstellen ist. Bei den Arbeiten am SMART-Projekt wurden heuristische Formeln zur Berechnung der Indexierungsgewichte für Dokumente (und Fragen) entwickelt, die sich als besonders leistungsfähig erwiesen haben. Diese Formeln wurden später im Rahmen der Arbeiten zu den experimentellen Systemen Inquery (U. Massachusetts / Bruce Croft) und OKAPI (MS Research Lab Cambridge / Stephen Robertson) weiterentwickelt. Wir stellen hier eine relativ neue Variante der Gewichtungformel vor.

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

- d_m^T Menge der in d_m vorkommenden Terms
- l_m Dokumentlänge (# Anzahl laufende Wörter in d_m)
- al durchschnittliche Dokumentlänge in \underline{D}
- tf_{mi} : Vorkommenshäufigkeit (Vkh) von t_i in d_m .
- n_i : # Dokumente, in denen t_i vorkommt.
- $|\underline{D}|$: # 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:

$$idf_i = \frac{\log \frac{|\underline{D}|}{n_i}}{|\underline{D}| + 1} \quad (5.10)$$

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

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, geht diese ebenfalls mit ein, und zwar als Verhältnis zur durchschnittlichen Dokumentlänge in der Kollektion:

$$ntf_i = \frac{tf_{mi}}{tf_{mi} + 0.5 + 1.5 \frac{l_m}{al}} \quad (5.11)$$

Das endgültige Indexierungsgewicht ergibt sich als Produkt der beiden Komponenten und wird daher meist als tfidf-Gewichtung bezeichnet:

$$w_{mi} = ntf_i \cdot idf_i \quad (5.12)$$

Tabelle 5.3 zeigt einige experimentelle Ergebnisse (aus [Salton & Buckley 88] mit einer früheren Version der tfidf-Formel aus dem SMART-Projekt) zu dieser Art der Gewichtung im Vergleich zu einer rein binären Gewichtung (Coordination Level Match). Dabei wurden die Gewichtungsformeln 5.10–5.12 sowohl zur Dokumentindexierung als auch zur Bestimmung des Fragevektors angewendet.

5.5.4 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 Dokumentensammlung. 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 $|D|$.
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 Ähnlichkeitmatrix gibt es eine ganze Reihe von Cluster-Algorithmen (siehe die einschlägige Literatur). Prinzipiell gibt es drei wesentliche Strategien: hierarchisches, agglomeratives und partitionierendes Clustering.

Agglomeratives Clustering geht von einem vorgegebenen Schwellenwert α für die Ähnlichkeit aus. Es erfolgt ein einmaliger Durchgang durch alle Dokumente mit dem Ziel, diese zu (anderen) Clustern hinzuzufügen. Entsprechend der jeweiligen Bedingung für die Aufnahme eines Dokumentes d_k in ein Cluster C_l kann man unter anderem folgende Verfahren unterscheiden:

- a) single link-Clustering:

$$\alpha \leq \min_{d_i \in C_l} \text{sim}(d_k, d_i)$$

- b) complete link-Clustering:

$$\alpha \leq \max_{d_i \in C_l} \text{sim}(d_k, d_i)$$

- c) average link-Clustering:

$$\alpha \leq \frac{1}{|C_l|} \sum_{d_i \in C_l} \text{sim}(d_k, d_i)$$

Gibt es kein solches Cluster, wird für d_k ein neues Cluster gebildet. Der Aufwand für all diese Verfahren beträgt (aufgrund der vollständigen Berechnung der Ähnlichkeitmatrix) $O(n^2)$.

Partitionierendes Clustering begnügt sich dagegen mit einem Aufwand von $O(kn)$. Dabei ist k die vorzugebende Anzahl zu bildender Cluster. Zu Beginn bestimmt das Verfahren k "seed"-Dokumente, die hinreichend unterschiedlich sind. Diese bilden jeweils den Kern eines der Cluster C_1, \dots, C_k . Alle übrigen Dokumente werden dann jeweils dem ähnlichsten Cluster hinzugefügt. Die Qualität dieses Verfahrens hängt stark von der Wahl der seed-Dokumente ab, wofür es wiederum verschiedene Strategien gibt.

Hierarchisches Clustering erzeugt eine baumförmige Struktur von ineinander geschachtelten Clustern. Man kann diese Struktur entweder bottom up oder top down erzeugen:

- Bei der bottom-up-Vorgehensweise bildet zunächst jede Instanz ein eigenes Cluster. In jedem Schritt werden dann jeweils die beiden Cluster mit der kleinsten Instanz vereinigt; dies wird fortgesetzt, bis nur noch ein Cluster übrig ist — die Wurzel des Baumes. Die Distanz zwischen zwei Clustern kann ähnlich wie beim agglomerativen Clustering unterschiedlich definiert werden — z.B. als kleinster/größter Abstand zwischen zwei Instanzen, oder als Distanz der Zentroiden.

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.

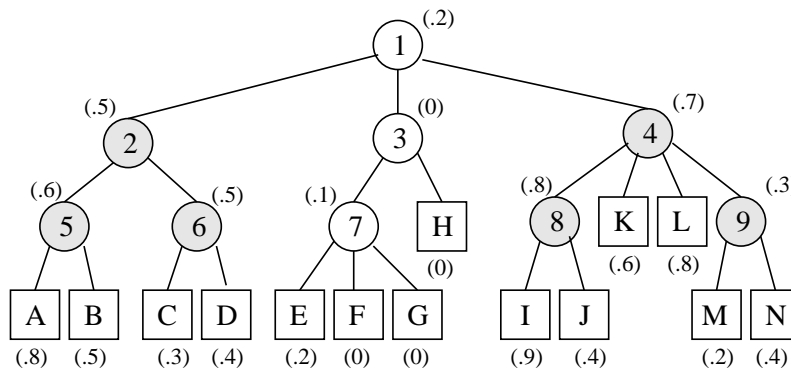


Abbildung 5.7: Beispiel zum Cluster-Retrieval

Abbildung 5.7 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.
- + 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 Dokumentenvektor 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 Probabilistisches Clustering

Während die bisher vorgestellten Cluster-Verfahren die Dokumentmenge jeweils in disjunkte Cluster zerlegen, strebt probabilistisches Clustering eine unscharfe Aufteilung an. Dabei gehört ein Dokument mit einer gewissen Wahrscheinlichkeit zu einem Cluster¹. Im Folgenden bezeichne C^1, \dots, C^k die k Cluster. Der Einfachheit halber gehen wir von einer binären Dokumentindexierung aus, so dass ein Dokument d_m durch einen Vektor \mathbf{x}_m repräsentiert wird mit $x_{m_i} = 1$, falls $t_i \in d_m^T$, und $x_{m_i} = 0$ sonst. Gesucht ist nun die Wahrscheinlichkeit $P(C^j|\mathbf{x})$ dass das Dokument mit Vektor \mathbf{x} zum Cluster C^j gehört. Dabei gelte, dass die Summe über alle Cluster 1 ist: $\sum_{j=1}^k P(C^j|\mathbf{x}) = 1$.

Um diese Wahrscheinlichkeit zu berechnen, wenden wir zunächst das Bayes'sche Theorem an:

$$P(a|b) = \frac{P(a, b)}{P(b)} = \frac{P(b|a) \cdot P(a)}{P(b)}$$

¹Einen ähnlichen Ansatz verfolgt auch Fuzzy-Clustering

Damit erhalten wir

$$P(C^j|\mathbf{x}) = \frac{P(\mathbf{x}|C^j)P(C^j)}{P(\mathbf{x})} \quad (5.13)$$

$$= \frac{P(\mathbf{x}|C^j)P(C^j)}{\sum_{l=1}^k P(C^l)P(\mathbf{x}|C^l)} \quad (5.14)$$

Hierbei ist $P(C^j)$ die Wahrscheinlichkeit, dass ein beliebiges Dokument zum Cluster C^j gehört, und $P(\mathbf{x}|C^j)$ gibt die Wahrscheinlichkeit an, dass ein zufälliges Dokument aus Cluster C^j den Merkmalsvektor \mathbf{x} hat. Um letztere zu berechnen, nehmen wir an, dass die einzelnen Merkmale (d.h. die Termvorkommen) im Cluster C^j unabhängig voneinander verteilt sind:

$$P(\mathbf{x}|C^j) = \prod_i P(x_i|C^j) \quad (5.15)$$

$$= \prod_{x_i=1} P(x_i = 1|C^j) \cdot \prod_{x_i=0} P(x_i = 0|C^j) \quad (5.16)$$

Da es sich um binäre Merkmale handelt, konnten wir in (5.16) nach dem Vorkommen/Nicht-Vorkommen der einzelnen Terme aufspalten. Hierbei ist $P(x_i = 1|C^j)$ die Wahrscheinlichkeit, dass der Term t_i in einem zufälligen Dokument des Clusters C^j vorkommt. Bezeichnen wir diese Wahrscheinlichkeit mit q_i^j , so können wir (5.16) umschreiben zu

$$P(\mathbf{x}|C^j) = \prod_{x_i=1} q_i^j \cdot \prod_{x_i=0} (1 - q_i^j) \quad (5.17)$$

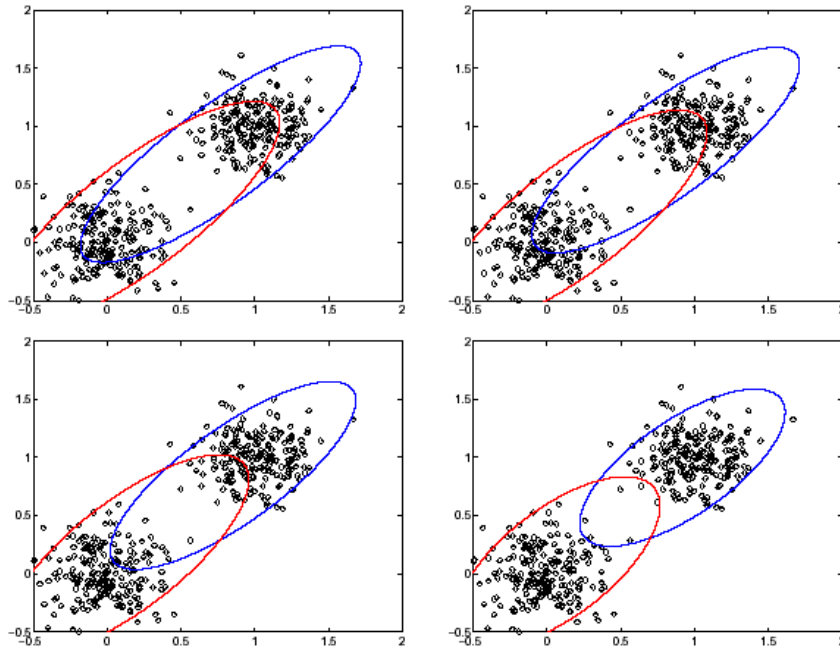


Abbildung 5.8: Probabilistisches Clustering: Iterationen bei 2 normalverteilten Merkmalen

Um das Modell anzuwenden, müssen wir die Parameter $p_j := P(C^j)$ und q_i^j schätzen. Dabei stellt sich das Problem, dass die Cluster nicht von vornherein bekannt sind. Diese Schwierigkeit lässt sich aber durch Anwendung des EM-Algorithmus' (expectation maximization) überwinden, der aus zwei Schritten besteht:

1. E: Berechne die Cluster-Wahrscheinlichkeit für jede Instanz
 2. M: Schätze die Parameter basierend auf den Cluster-Wahrscheinlichkeiten
- Zur Parameterschätzung verwenden wir folgende Formeln:

$$n^j = \sum_{d_m \in D} P(C^j | \mathbf{x}_m) \quad (5.18)$$

$$p^j = \frac{n^j}{|D|} \quad (5.19)$$

$$q_i^j \approx \frac{1}{n^j} \sum_{d_m \in D} x_{m_i} \cdot P(C^j | \mathbf{x}_m) \quad (5.20)$$

$$\approx \frac{1}{n^j + 1} \left(p^j + \sum_{d_m \in D} x_{m_i} \cdot P(C^j | \mathbf{x}_m) \right) \quad (5.21)$$

Dabei wird in (5.21) ein sogenannter Bayes'scher Schätzer verwendet (siehe nächstes Kapitel), der insbesondere Nullwerte bei den Wahrscheinlichkeitsschätzungen vermeidet.

Die Anwendung des Verfahrens läuft dann wie folgt ab:

1. Wähle die Anzahl k der zu bildender Cluster.
2. Bestimme k "seed"-Dokumente, die hinreichend unterschiedlich sind. Diese bilden jeweils den Kern eines der Cluster C^1, \dots, C^k
3. Initialisierung der Parameter: Setze $n^j = 1$ und $p^j = 1/k$. Ferner sei

$$P(C^j | \mathbf{x}_m) = \begin{cases} 1, & \text{falls } d_m \text{ seed von } C^j \\ 0, & \text{sonst} \end{cases}$$

Berechne daraus initiale Werte für die q_i^j .

4. Für alle Dokumente $d_m \in D$: Berechne $P(C^j | \mathbf{x}_m)$ für $j = 1 \dots, k$.
5. Berechne neue Parameter n^j , p^j und q_i^j .
6. Wiederhole die letzten beiden Schritte, bis die Cluster stabil sind.

Das Verfahren lässt sich auch leicht auf numerische Merkmale erweitern — die meisten Clustering-Verfahren wurden ursprünglich für numerische Merkmale entwickelt. Beim probabilistischen Clustering nimmt man für solche Merkmale x_i eine Normalverteilung an:

$$P(x_i | C^j) = \frac{1}{\sqrt{1\pi\sigma_i}} e^{-\frac{(x_i - \mu_i^j)^2}{2(\sigma_i^j)^2}} \quad (5.22)$$

Anstelle einer einzelnen Wahrscheinlichkeit q_i^j muss man in diesem Fall Mittelwert und Varianz der zugehörigen Normalverteilung schätzen:

$$\mu_i^j = \frac{1}{n^j} \sum_{d_m \in D} x_{m_i} \cdot P(C^j | \mathbf{x}_m) \quad (5.23)$$

$$\sigma_i^j = \frac{1}{n^j} \sum_{d_m \in D} (x_{m_i} - \mu_i^j)^2 \cdot P(C^j | \mathbf{x}_m) \quad (5.24)$$

5.6.4 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. 5.7), 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.

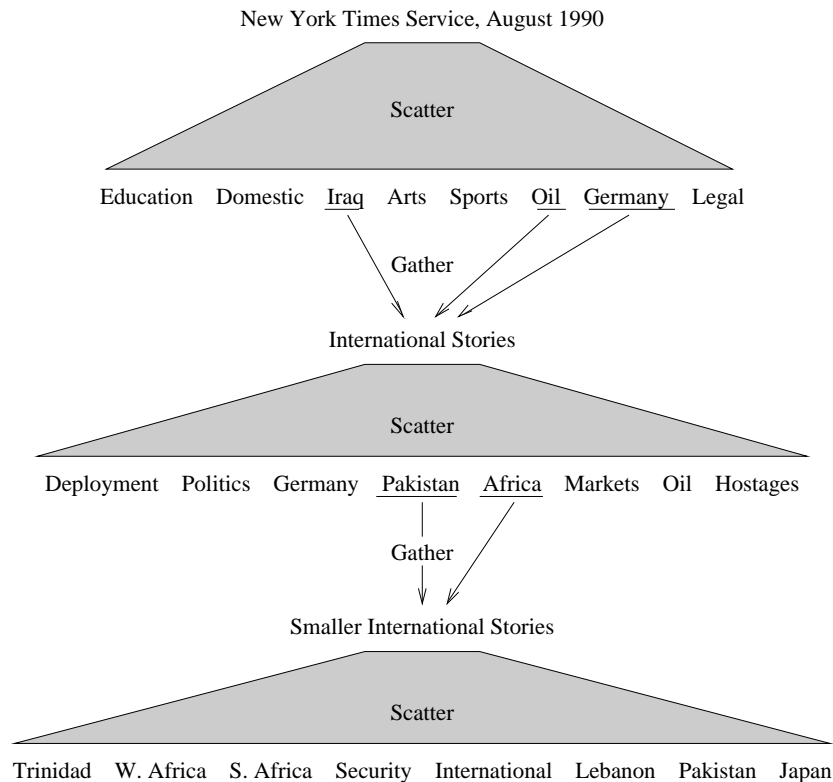


Abbildung 5.9: Beispiel zu Scatter/Gather

5.6.5 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 (mittels partitionierendem Clustering) und dem Benutzer Repräsentanten für die resultierenden Cluster angezeigt. Diese Repräsentanten bestehen dabei einerseits aus Titeln von Dokumenten in der Nähe des Cluster-Zentroiden, andererseits aus häufig im Cluster vorkommenden Wörtern.

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 5.9 zeigt eine Folge von drei Suchschritten. Scatter/Gather-Browsing kann somit als Kombination von Browsing in (statischen) Clustern und Relevance Feedback aufgefasst werden.

<input type="checkbox"/>	Cluster 1 Size: 4 assistant director deputy secretary special affair division administrator management staff po
<input type="radio"/>	603252 "Excepted Service; Consolidated Listing of Schedules A, B, and C Exceptions"
<input type="radio"/>	329912 "Excepted Service; Consolidated Listing of Schedules A, B, and C Exceptions"
<input type="radio"/>	610814 "5 CFR Part 737"
<input type="radio"/>	317319 "SES Positions That Were Career Reserved During 1988"
<input type="checkbox"/>	Cluster 2 Size: 187 deposit capital asset insurance risk fail save credit rate market account billion
<input type="radio"/>	631435 "World Business (A Special Report): Eastern Europe --- The Idea Man: France's Jacques Attali Is the Driving Force Behi
<input type="radio"/>	658624 "Politics & Policy: CIA Warned In '86 of Entry Of BCCI to U.S. ---- By Peter Truell Staff Reporter of The Wall Street Jour
<input type="radio"/>	39340 "House, Senate Versions Compared"
<input type="radio"/>	402897 "Under Fire: World Bank's Conable Runs Into Criticism On Poor Nations' Debt --- Liberals Assail His Refusal To Give M
<input type="radio"/>	333197 "Federal Reserve Bank Services"
<input type="checkbox"/>	Cluster 3 Size: 217 section information 2 requirement regulation 3 request rule record 5 provision procedure
<input type="radio"/>	690665 "Security is big business. (balancing security systems and user training to achieve data security)"
<input type="radio"/>	592791 "Organization; Farm Credit System Financial Assistance Corp."
<input type="radio"/>	322941 "PART 78 EDUCATION APPEAL BOARD"
<input type="radio"/>	334160 "12 CFR Parts 7 and 32"
<input type="radio"/>	334479 "Privacy Act of 1974; Systems of Records"
<input type="checkbox"/>	Cluster 4 Size: 85 investigation allege fraud court lawyer firm prosecutor jury bcci american grand defendant
<input type="radio"/>	631459 "The Safra Affair: A Saga of Corporate Intrigue --- The Vendetta: How American Express Orchestrated a Smear Of Rival E
<input type="radio"/>	662803 "Kidder Advised U.S. It Was Helping BCCI Buy an Interest in First American ---- By Peter Truell Staff Reporter of The W
<input type="radio"/>	21620 "High Court Refuses to Dismiss Helmsley Indictment"
<input type="radio"/>	649610 "The Americas: Peru: Another Link in the BCCI Money Laundering Chain? ---- By Alvaro Vargas Llosa"
<input type="radio"/>	572658 "Senior Banker Charged In Money Laundering Operation"
<input type="checkbox"/>	Cluster 5 Size: 7 marcos philippine marcoses unite order export respondent racketeering khashoggi buy man
<input type="radio"/>	80628 "Former Interior Minister Extradited to Miami on Drug Charges"
<input type="radio"/>	37937 "Prosecutors Seek Judgment Against Marcos Even in Event of Death"
<input type="radio"/>	328041 "Action Affecting Export Privileges; Marek Cieslak"
<input type="radio"/>	575028 "Federal Grand Jury Indicts Marcos"

Abbildung 5.10: Scatter/Gather: Bildschirmausgabe

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. 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.) 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 $T = \{t_1, \dots, t_n\}$ 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 $\vec{x} = (x_1, \dots, x_n)$ with $x_i = 1$, if $t_i \in d_m^T$ 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 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

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

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

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

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

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

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 occurring in the set q_k^T of query terms. With these notations and simplifications, we arrive at the formula

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

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

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

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

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 $t_i \in q_k^T$. 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, 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).

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

\vec{x}	$P(R q, \vec{x})$	
	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.

Here d_m and 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

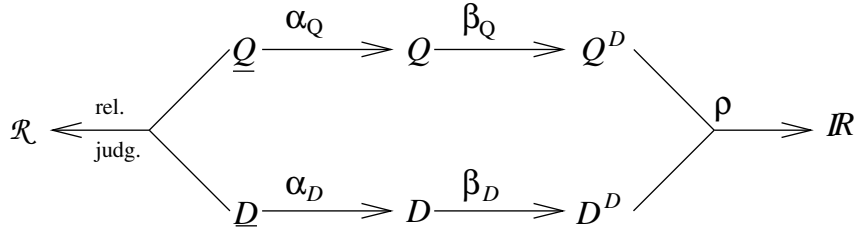


Figure 6.1: Conceptual model

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 $q_k^J = \{(d_m, r(d_m, q_k))\}$. 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 explicitly 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 $\rho(q_k^D, d_m^D)$ 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).

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 $Q_L \times D_L \times T_L$ of $Q \times D \times T$ (where T denotes the set of terms in the collection) from which we can derive probabilistic parameters. These parameters can be used in the application phase for the improvement of the descriptions of documents and queries.

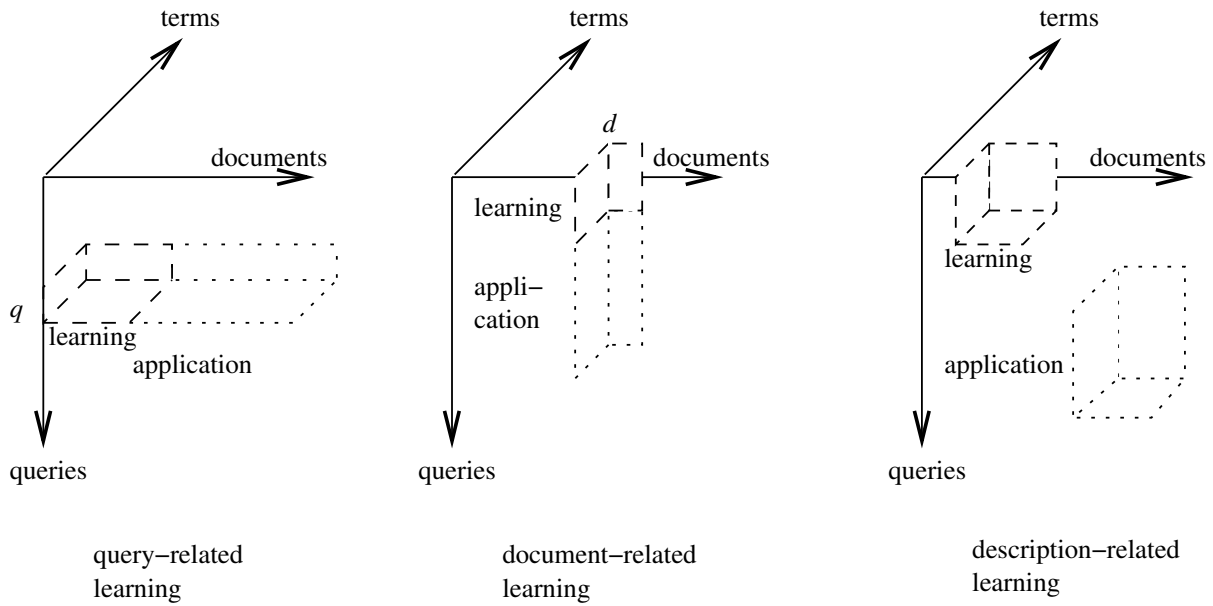


Figure 6.2: Learning approaches in IR

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 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 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 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 these 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 $Q_L \times D_L \times T_L$. For the application phase, there are no restrictions concerning the subset $Q_A \times D_A \times T_A$ 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.

The event space is $Q \times D$. A single element of this event space is a query-document pair (q_k, d_m) ,

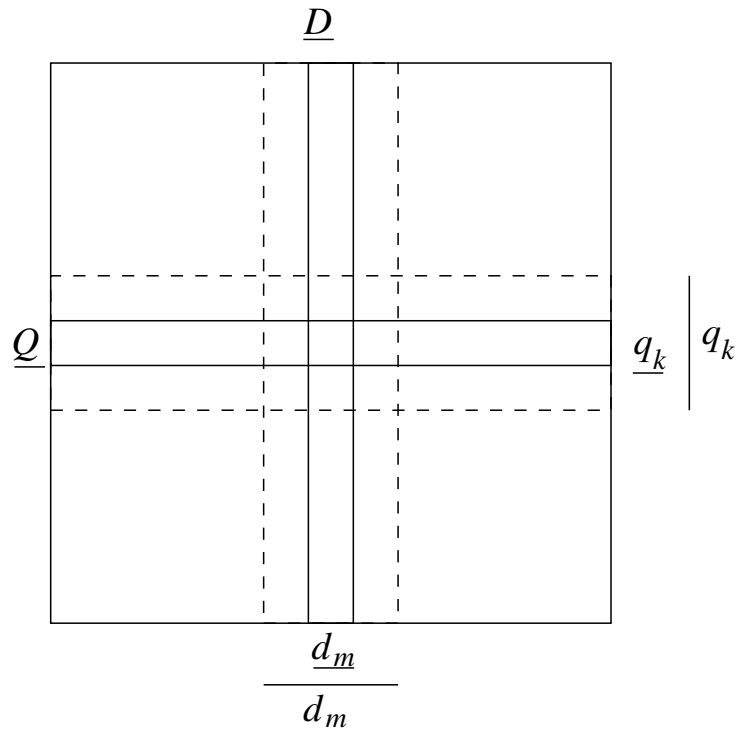


Figure 6.3: Event space of relevance models

where we assume that all these elements are equiprobable. Associated with each element is a relevance judgement $r(\underline{d}_m, \underline{q}_k) \in \mathcal{R}$. 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).

6.2.5.1 Decision-theoretic justification of the PRP

The decision-theoretic justification of the PRP is as follows (see [Robertson 77]): Let \bar{C} denote the costs for the retrieval of a nonrelevant document, and C the costs for the retrieval of a relevant document. Since a user prefers relevant documents, we assume that $\bar{C} > C$. Then the *expected costs* for retrieving a document d are computed as

$$EC(d) = C \cdot P(R|q, d) + \bar{C} \cdot (1 - P(R|q, d))$$

In response to a query, a user looks at output documents in the ranked order and stops at an arbitrary point. Thus, the total costs of retrieval can be computed as follows: Let us assume that the ranking function $r(i)$ determines the index of the document in the database to be placed at rank i for the current query. Then the costs for retrieving l documents are

$$\begin{aligned} EC(q, l) &= EC(q, d_{r(1)}, d_{r(2)}, \dots, d_{r(l)}) \\ &= \sum_{i=1}^l EC(q, d_{r(i)}) \end{aligned}$$

In order to minimize the sum of expected costs at any cutoff point, documents have to be ranked according to increasing expected costs, i.e. $EC(q, d_{r(i)}) \leq EC(q, d_{r(i+1)})$ for $i = 1, \dots, l - 1$. Because of $C < \bar{C}$, this condition is equivalent to

$$P(R|q, d_{r(i)}) \geq P(R|q, d_{r(i+1)}).$$

So we have the rule that documents should be ranked according to their decreasing probability of being relevant, in order to minimize expected costs of retrieval. Thus, probabilistic retrieval models are directly related to retrieval quality — a major advantage over other models for which such a claim cannot be made.

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 $R_1 < R_2 < \dots < R_n$ the corresponding costs for the retrieval of a document with that retrieval judgement are C_1, C_2, \dots, C_n . Then documents should be ranked according to their expected costs

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

In contrast to the binary case where only the probability $P(R|q, d_m)$ has to be estimated for a query-document pair, here $n - 1$ estimates $P(R_l|q, d_m)$ 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 $r \in [0, 1]$. In this case, the probability distribution $P(R_l|q, d_m)$ from above is replaced by a density function $p(r|q, d_m)$ 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.

6.2.5.2 Justifications of the PRP based on effectiveness measures

For any two events a, b , Bayes' theorem yields (see derivation of BIR model):

$$\begin{aligned}\frac{P(a|b)}{P(\bar{a}|b)} &= \frac{P(b|a)P(a)}{P(b|\bar{a})P(\bar{a})} \\ \log \frac{P(a|b)}{P(\bar{a}|b)} &= \log \frac{P(b|a)}{P(b|\bar{a})} + \log \frac{P(a)}{P(\bar{a})} \\ \text{logit } P(a|b) &= \log \frac{P(b|a)}{P(b|\bar{a})} + \text{logit } P(a)\end{aligned}\quad (6.5)$$

Here $\text{logit } P(x)$ is defined as

$$\text{logit } P(x) = \log \frac{P(x)}{P(\bar{x})}.$$

For a given query q , we assume that the system has produced a ranked list of documents, and that the user starts looking at the documents from the beginning of this list, up to a certain point. Let S denote the set of documents the user has seen by following this procedure.

Then we can define the following parameters, which are probabilistic interpretations of the corresponding effectiveness measures:

$$\begin{aligned}P(S|R, q) &= P(\text{doc. retrieved}|\text{doc. relevant}) && (\text{recall}) \\ P(S|\bar{R}, q) &= P(\text{doc. retrieved}|\text{doc. nonrelevant}) && (\text{fallout}) \\ P(R|S, q) &= P(\text{doc. relevant}|\text{doc. retrieved}) && (\text{precision}) \\ P(R|q) &= P(\text{doc. relevant}) && (\text{generality})\end{aligned}$$

In a similar way, we define the following probabilistic parameters for a single document $d_i \in S$ of the retrieved set.

$$\begin{aligned}P(d_i|R, q) &= P(\text{doc. is } d_i|\text{doc. relevant}) \\ P(d_i|\bar{R}, q) &= P(\text{doc. is } d_i|\text{doc. nonrelevant}) \\ P(R|q, d_i) &= P(\text{doc. relevant}|\text{doc. is } d_i)\end{aligned}$$

The last parameter $P(R|q, d_i)$ is the probability of relevance, which we will use for ranking documents.

Based on these parameters defined for single documents, we can compute recall and fallout as follows:

$$P(S|R, q) = \sum_{d_i \in S} P(d_i|R, q) \quad (6.6)$$

$$P(S|\bar{R}, q) = \sum_{d_i \in S} P(d_i|\bar{R}, q) \quad (6.7)$$

In order to justify the PRP based on effectiveness measures, we assume that the stopping criterion of the user is defined wrt. to one of the effectiveness measures; then we can show that the values for the other measures are optimized by the PRP. For this purpose, we apply the transformations following Bayes' theorem from above (eqn 6.5):

$$\text{logit } P(R|q, d_i) = \log \frac{P(d_i|R, q)}{P(d_i|\bar{R}, q)} + \text{logit } P(R|q) \quad (6.8)$$

$$P(d_i|R, q) = x_i \cdot P(d_i|\bar{R}, q) \quad (6.9)$$

$$\text{with } x_i = \exp(\text{logit } P(R|q, d_i) - \text{logit } P(R|q))$$

So x_i is a monotonic function of $P(R|q, d_i)$, the probability of relevance

- For a given cutoff defined by $P(S|\bar{R}, q)$ (fallout), eqn 6.7 in combination with eqn 6.8 tells us that we can maximize $P(S|R, q)$ (expected recall) by including documents with highest values of $P(R|q, d_i)$ in the retrieved set, i.e. rank documents according to probability of relevance.
- Vice versa, if we choose recall (the number of relevant documents seen) as stopping criterion, then the PRP yields minimum fallout
- For a cutoff defined in terms of the number of documents retrieved, we can maximize expected recall (eqn 6.6) and minimize expected fallout (eqn 6.7) by choosing the documents with the highest ratio $P(d_i|R, q)/P(d_i|\bar{R}, q)$, i.e. follow the PRP according to eqn 6.9.

Applying the transformation 6.5 to the fallout formula

$$\text{logit } P(R|S, q) = \log \frac{P(S|R, q)}{P(S|\bar{R}, q)} + \text{logit } P(R|q),$$

we can see that expected precision is minimized for any of the stopping criteria mentioned before (given recall, fallout, or number of documents retrieved).

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 (q_k, d_m) , and the classification is the assignment of a value R_l from a relevance scale $\mathcal{R} = (R_1, \dots, R_n)$.

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 $P(R_l|\vec{x}(q_k, d_m))$, $l = 1, \dots, n$ for a pair (q_k, 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	$\log(\# \text{ descriptors common to query and document})$
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	$\log \sum (\text{indexing weights of common descriptors})$
x_9	$\log(\# \text{ descriptors in the query})$

Tabelle 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 $r(q_k, d_m)$ of a request-document pair is represented by a vector \vec{y} with $y_l = 1$, if $r(q_k, d_m) = R_l$, and $y_l = 0$ otherwise.

Now we seek for a regression function $\vec{e}_{opt}(\vec{x})$ which yields an optimum approximation \hat{y} of the class variable \vec{y} . As optimizing criterion, we use minimum squared errors here ($E(\cdot)$ denotes the expectation):

$$E(|\vec{y} - \vec{e}_{opt}(\vec{x})|^2) \stackrel{!}{=} \min. \quad (6.10)$$

With this condition, $\vec{e}_{opt}(\vec{x})$ yields an estimate of the probabilities $P(R_l|\vec{x})$, $l = 1, \dots, n$ (see [Schürmann 77, pp. 163–164]).

Equation 6.10 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

$$E(|E(\vec{y}|\vec{x}) - \hat{\vec{y}}|^2) \stackrel{!}{=} \min. \quad (6.11)$$

So our restricted optimization 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 $\vec{v}(\vec{x}) = (v_1, \dots, v_L)$ has to be defined as

$$\vec{v}(\vec{x}) = (1, x_1, x_2, \dots, x_N, x_1^2, x_1x_2, \dots)$$

where N is the number of dimensions of \vec{x} . The class of polynomials is given by the components $x_i^l \cdot x_j^m \cdot x_k^n \cdot \dots$ ($i, j, k, \dots \in [1, N]; l, m, n, \dots \geq 0$) 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 $i=1, \dots, L; l=1, \dots, n$ is the coefficient matrix which has to be estimated. So $P(R_l|\vec{x})$ is approximated by the polynomial

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

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

$$E(\vec{v} \cdot \vec{v}^T) \cdot A = E(\vec{v} \cdot \vec{y}^T). \quad (6.12)$$

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 $M = (\overline{\vec{v} \cdot \vec{v}^T} \cdot \overline{\vec{y} \cdot \vec{y}^T})$ which contains both sides of the equation system 6.12. 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 $l=1, \dots, L; j=1, \dots, L+n$ denote the matrix M before the i th 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:

$$d_j^{(i)} = \frac{1}{m_{jj}^{(i)^2}} \cdot \sum_{l=L+1}^{L+n} m_{jl}^{(i)^2}. \quad (6.13)$$

After each iteration, this procedure yields a preliminary solution: With the i th 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).

\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

Tabelle 6.4: Example for the LSP approach

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 6.13: $d_1^{(1)} = 0.50$, $d_2^{(1)} = 0.52$ and $d_3^{(1)} = 0.50$. So we have 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

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

$$e_1^{(3)} = 0.17 + 0.33x_1 + 0.17x_2 \text{ and } e_2^{(3)} = 0.83 - 0.33x_1 - 0.17x_2.$$

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 $P(R_i|\vec{x})$.

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 (q_k, 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), 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 $\vec{z}_k = (z_{k_1}, \dots, z_{k_n})$ 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

$$P(R|\vec{z}_k, d_m) = P(R|d_m) \cdot \frac{P(\vec{z}_k|R, d_m)}{P(\vec{z}_k|d_m)} \quad (6.14)$$

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 6.14 can be transformed into

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

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:

$$\begin{aligned}
 P(R|\vec{z}_k, d_m) &= \frac{\prod_{i=1}^n P(z_{k_i})}{P(\vec{z}_k)} \cdot P(R|d_m) \cdot \prod_{i=1}^n \frac{P(R|z_{k_i}, d_m)}{P(R|d_m)} \\
 &= \frac{\prod_{i=1}^n P(z_{k_i})}{P(\vec{z}_k)} \cdot P(R|d_m) \cdot \prod_{z_{k_i}=1} \frac{P(R|z_{k_i}=1, d_m)}{P(R|d_m)} \\
 &\quad \cdot \prod_{z_{k_i}=0} \frac{P(R|z_{k_i}=0, d_m)}{P(R|d_m)}
 \end{aligned} \tag{6.15}$$

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 6.15 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

$$P(R|q_k, d_m) = c_k \cdot P(R|d_m) \cdot \prod_{t_i \in q_k^T \cap d_m^T} \frac{P(R|t_i, d_m)}{P(R|d_m)}. \tag{6.16}$$

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

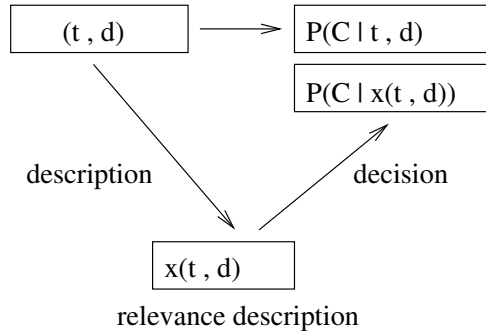


Abbildung 6.4: Subdivision of the indexing task in a description step and a decision step

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 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

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	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)

Tabelle 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

Tabelle 6.6: Probability estimates for the example from table 6.5

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 $L \subset Q \times D \times \mathcal{R}$ of query-document pairs for which we have relevance judgements, so $L = \{(q_k, d_m, r_{km})\}$. 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 $L^x = [(x(t_i, d_m), r_{km}) | t_i \in q_k^T \cap d_m^T \wedge (q_k, d_m, r_{km}) \in L]$. From this set with multiple occurrences of elements, the parameters $P(R|x(t_i, d_m))$ could be estimated directly 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 derived¹.

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 5.5.3), 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

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

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 $P(d_m \in K_{ij} | tf_{im}=l)$ 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

¹As 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.

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 $q_k^J = \{(d_m, r(d_m, q_k))\}$; 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 $(d_{m_1}, \dots, d_{m_n})^T$, where d_{m_i} is the relevance description of t_i w.r.t. d_m , and \vec{c}_m is the binary vector $(c_{m_1}, \dots, c_{m_n})^T$ 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 $\vec{x} = (x_1, \dots, x_n)$ denote a set of relevance descriptions for the index terms t_1, \dots, 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

$$O(R|q_k, \vec{x}) = O(R|q_k) \frac{P(\vec{x}|\bar{R}, q_k)}{P(\vec{x}|R, q_k)}. \quad (6.17)$$

The linked dependence assumption

$$\frac{P(\vec{x}|R, q_k)}{P(\vec{x}|\bar{R}, q_k)} = \prod_{i=1}^n \frac{P(x_i|R, q_k)}{P(x_i|\bar{R}, q_k)} \quad (6.18)$$

yields

$$O(R|q_k, \vec{x}) = O(R|q_k) \prod_{i=1}^n \frac{P(x_i|R, q_k)}{P(x_i|\bar{R}, q_k)}. \quad (6.19)$$

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

$$O(R|q_k, \vec{x}) = O(R|q_k) \prod_{i=1}^n \frac{P(x_i|C_i) \cdot P(C_i|R, q_k) + P(x_i|\bar{C}_i) \cdot P(\bar{C}_i|R, q_k)}{P(x_i|C_i) \cdot P(C_i|\bar{R}, q_k) + P(x_i|\bar{C}_i) \cdot P(\bar{C}_i|\bar{R}, q_k)} \quad (6.20)$$

$$= O(R|q_k) \prod_{i=1}^n \frac{\frac{P(C_i|x_i)}{P(C_i)} \cdot P(C_i|R, q_k) + \frac{P(\bar{C}_i|x_i)}{P(\bar{C}_i)} \cdot P(\bar{C}_i|R, q_k)}{\frac{P(C_i|x_i)}{P(C_i)} \cdot P(C_i|\bar{R}, q_k) + \frac{P(\bar{C}_i|x_i)}{P(\bar{C}_i)} \cdot P(\bar{C}_i|\bar{R}, q_k)} \quad (6.21)$$

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

$$O(R|q_k, \vec{x}=\vec{d}_m) = O(R|q_k) \prod_{t_i \in q_k^T} \frac{\frac{u_{m_i}}{q_i} p_{ik} + \frac{1-u_{m_i}}{1-q_i} (1-p_{ik})}{\frac{u_{m_i}}{q_i} r_{ik} + \frac{1-u_{m_i}}{1-q_i} (1-r_{ik})}. \quad (6.22)$$

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

$$O(R|q_k, \vec{x}=\vec{d}_m) \approx O(R|q_k) \prod_{t_i \in q_k^T} \frac{u_{m_i}}{q_i} p_{ik} + \frac{1-u_{m_i}}{1-q_i} (1-p_{ik}). \quad (6.23)$$

The parameters for the RPI formulas can be estimated by means of relevance feedback: let D_k^R denote the set of documents judged relevant w.r.t. q_k and 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 r_{ik} as the expectation for the nonrelevant documents:

$$\begin{aligned} q_i &= \frac{1}{|D|} \sum_{d_j \in D} u_{j_i}, \\ p_{ik} &= \frac{1}{|D_k^R|} \sum_{d_j \in D_k^R} u_{j_i}, \\ r_{ik} &= \frac{1}{|D_k^N|} \sum_{d_j \in D_k^N} u_{j_i}. \end{aligned}$$

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. [Ullman 89]). 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 :

$$P(d \rightarrow q) = P(q|d) \quad (6.24)$$

Assume that we have a probability space where terms represent disjoint events, as shown in figure 6.5. Taking the classical logical approach, one would compute $P(d \rightarrow q)$ as $P(\neg d \vee q)$. Assuming an equal probability distribution over the terms, this would give us $P(d \vee \neg) = 5/6$ for the left-hand side of figure 6.5. However, the result would be the same when we either would add the term t_1 to the document or t_4 to the query. Since this is not reasonable, classical logic seems to be inappropriate for this task. Thus, Rijsbergen proposed to define $P(d \rightarrow q)$ as conditional probability $P(q|d)$, yielding the value $2/3$ in this example. As another example, 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:

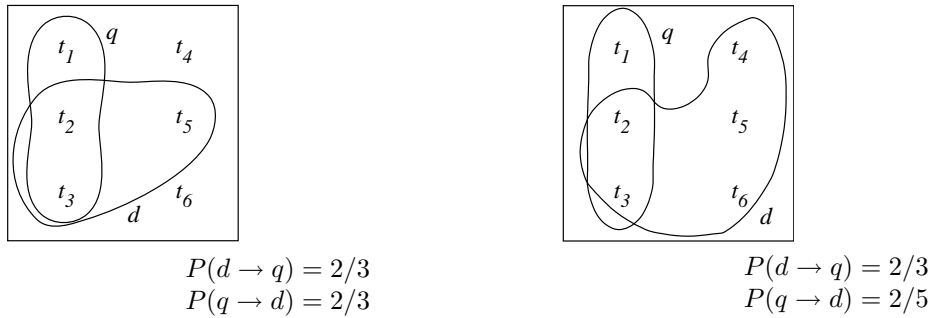
$$P(R|q, d) = P(R|q \leftarrow d)P(q \leftarrow d) + P(R|\neg(q \leftarrow d))P(\neg(q \leftarrow d)) \quad (6.25)$$

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 (6.25), $P(R)$ 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 besides 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.

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 left-hand case, a smaller fraction of the document is 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:

$$P(R|q, d) = f(P(d \rightarrow q), P(q \rightarrow d)) \quad (6.26)$$

Abbildung 6.5: $P(d \rightarrow q)$ vs. $P(q \rightarrow d)$

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

6.5 Parameter estimation

6.5.1 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.

6.5.2 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 triplet (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, \dots 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

$$g(q|x_k) = \frac{f(q) \cdot P(X=x_k|Q=q)}{\int_{-\infty}^{\infty} f(q) \cdot P(X=x_k|Q=q) dq} \quad (6.27)$$

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:

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

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

$$g(p|\frac{h}{f}) = \frac{p^{h+a-1} (1-p)^{f-h+b-1}}{B(h+a, f-h+b)} \quad (6.28)$$

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

$$L_1(\hat{p}, p_{ij}) = (\hat{p} - p_{ij})^2, \quad (6.29)$$

we also regard the loss function

$$L_2(p, \hat{p}) = \frac{(p - \hat{p})^2}{p(1-p)} \quad (6.30)$$

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 estimates²

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

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:

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

For the beta prior, we get

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

It is interesting to notice that the four different methods for deriving an estimate from the 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.

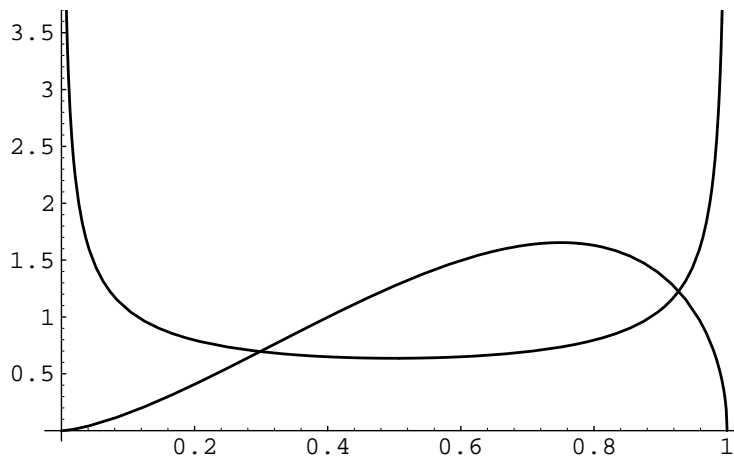


Abbildung 6.6: Prior (upper curve) and posterior distribution for $a = b = 0.5$, $f = 3$, $h = 2$

In order to illustrate this approach, we give some examples of prior and posterior distributions for p_{L_2} . Figure 6.6 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 6.7, 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 6.6) 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 6.8 shows prior and posterior distributions $f = 3$ and $h = 2$ and figure 6.9 gives the two posterior distributions for $f = 3$ and $h = 2$ and for $f' = 15$ and $h' = 10$, respectively.

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

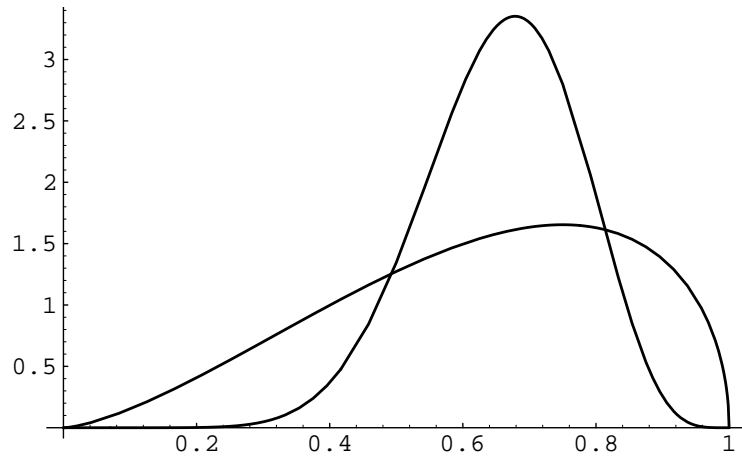


Abbildung 6.7: Posterior distributions for $a = b = 0.5$, $f = 3$, $h = 2$ and $f' = 15$, $h' = 10$

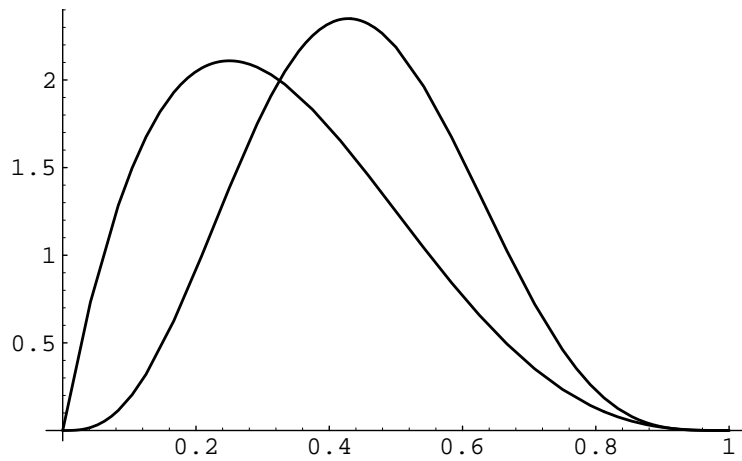


Abbildung 6.8: Prior (left curve) and posterior distribution for $a = 2$, $b = 4$, $f = 3$, $h = 2$

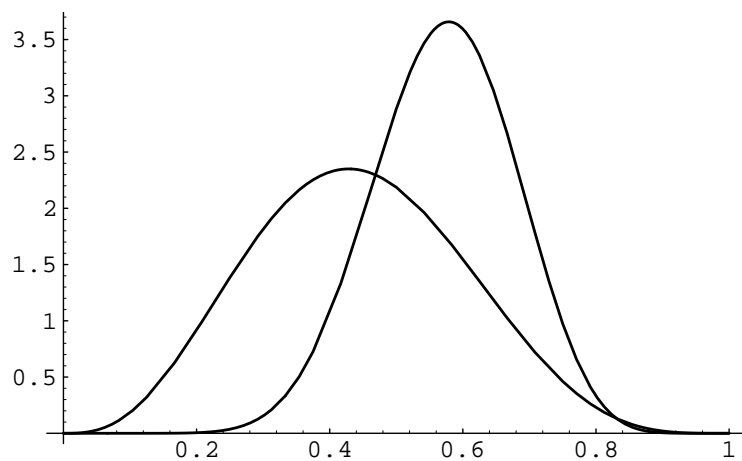


Abbildung 6.9: Posterior distributions for $a = 2$, $b = 4$, $f = 3$, $h = 2$ and $f' = 15$, $h' = 10$

6.5.3 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 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 6.29) is minimized:

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

$$\frac{dE}{d\hat{p}} = \sum_{(k,l)} 2\hat{p}p_l P(Z_g=(h, f)|Q=p_{kl}) - \sum_{(k,l)} 2p_{kl}p_l P(Z_g=(h, f)|Q=p_{kl}) \quad (6.31)$$

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

$$p_{opt} = \frac{\sum_{(k,l)} p_{kl}p_l P(Z_g=(h, f)|Q=p_{kl})}{\sum_{(k,l)} p_l P(Z_g=(h, f)|Q=p_{kl})} \quad (6.32)$$

$$(6.33)$$

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 6.32, 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

$$p_{opt}(e_i|e_j, (h, f, g)) = \frac{E^+(h, f, g)}{E^+(h, f, g) + E^-(h, f, g)} \quad (6.34)$$

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 $|Z(h, f, g)|$:

$$\begin{aligned}
E^+(h, f, g) &= \sum_{(k,l)} \binom{g}{f} p_l^f (1-p_l)^{g-f} \binom{f}{h} p_{kl}^h (1-p_{kl})^{f-h} \cdot p_l \cdot p_{kl} \\
&= \frac{h+1}{g+1} \sum_{(k,l)} \binom{g+1}{f+1} p_l^{f+1} (1-p_l)^{g+1-(f+1)} \cdot \\
&\quad \binom{f+1}{h+1} p_{kl}^{h+1} \cdot (1-p_{kl})^{f+1-(h+1)} \\
&= \frac{h+1}{g+1} E(h+1, f+1, g+1) \\
&\approx \frac{h+1}{g} E(h+1, f+1, g)
\end{aligned} \tag{6.35}$$

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

$$\begin{aligned}
E^-(h, f, g) &= \sum_{k,l} \binom{g}{f} p_l^f (1-p_l)^{g-f} \binom{f}{h} p_{kl}^h (1-p_{kl})^{f-h} p_l \cdot (1-p_{kl}) \\
&= \frac{f+1-h}{g+1} \sum_{k,l} \binom{g+1}{f+1} p_l^{f+1} (1-p_l)^{g+1-(f+1)} \cdot \\
&\quad \binom{f+1}{h} p_{kl}^h (1-p_{kl})^{f+1-h} \\
&= \frac{f+1-h}{g+1} E(h, f+1, g+1) \\
&\approx \frac{f+1-h}{g} E(h, f+1, g)
\end{aligned} \tag{6.36}$$

With these approximations for $E^+(h, f, g)$ and $E^-(h, f, g)$, we can estimate p_{opt} according to formula 6.34 as

$$p_{opt}(e_i|e_j, (h, f, g)) \approx \frac{(h+1) E(h+1, f+1, g)}{(h+1) E(h+1, f+1, g) + (f+1-h) \cdot E(h, f+1, g)} \tag{6.37}$$

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].

Chapter 7

Models based on propositional logic

In this chapter, we will show that based on the concept of uncertain inference, most classical retrieval models can be given a probabilistic interpretation. Most of the material presented here is based on the paper [Wong & Yao 95], which the reader should consult for further details. Surveys on probabilistic IR models are given in [Crestani et al. 98] and [Fuhr 92].

7.1 A Probabilistic Inference Model

Most text retrieval models represent documents as sets of (weighted) propositions. In order to set up a basic framework for these models, we assume a concept space U consisting of a set of elementary, disjoint concepts c_i (see figure 7.1).

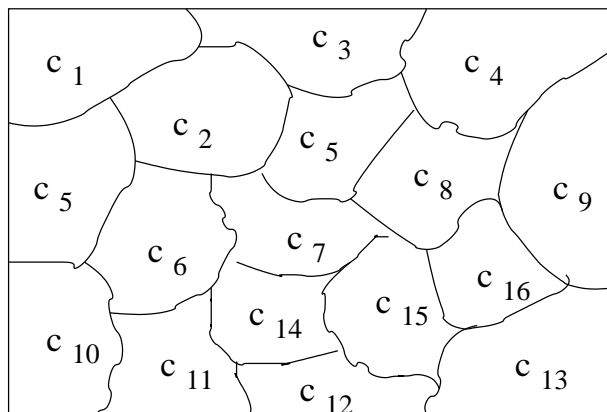


Figure 7.1: Concept space

Any proposition p is a set of concepts, i.e. a subset of the concept space ($p \subseteq U$). Boolean combinations of propositions can be expressed as set operations on this concept space. Let e.g. $p_1 = \{c_1, c_2, c_3\}$ and $p_2 = \{c_2, c_4\}$, then $p_1 \cap p_2 = \{c_2\}$.

In order to support probabilistic inference, we define a probability function $P(\cdot)$ over U , i.e.

$$\sum_{c_i \in U} P(c_i) = 1$$

Now queries and documents are treated as propositions as well, Considering the probability function,

we have

$$P(d) = \sum_{c_i \in d} P(c_i)$$

$$P(q \cap d) = \sum_{c_i \in q \cap d} P(c_i)$$

$$P(d \rightarrow q) = P(q|d) = \frac{P(q \cap d)}{P(d)}$$

7.2 Classical IR models

Now we will describe a number of classical IR models and show how they can be interpreted in terms of probabilistic inference. Whereas text retrieval is based on terms, our basic model uses concepts as elementary propositions; thus, we have to define the relationship between terms and concepts. A straightforward approach identifies each term with a concept (section 7.2.1). Alternatively, one can assume that terms are overlapping, so we need a different mapping from terms onto concepts (see section 7.2.2). Figure 7.2 gives a systematic survey of the classical IR models described in the following.

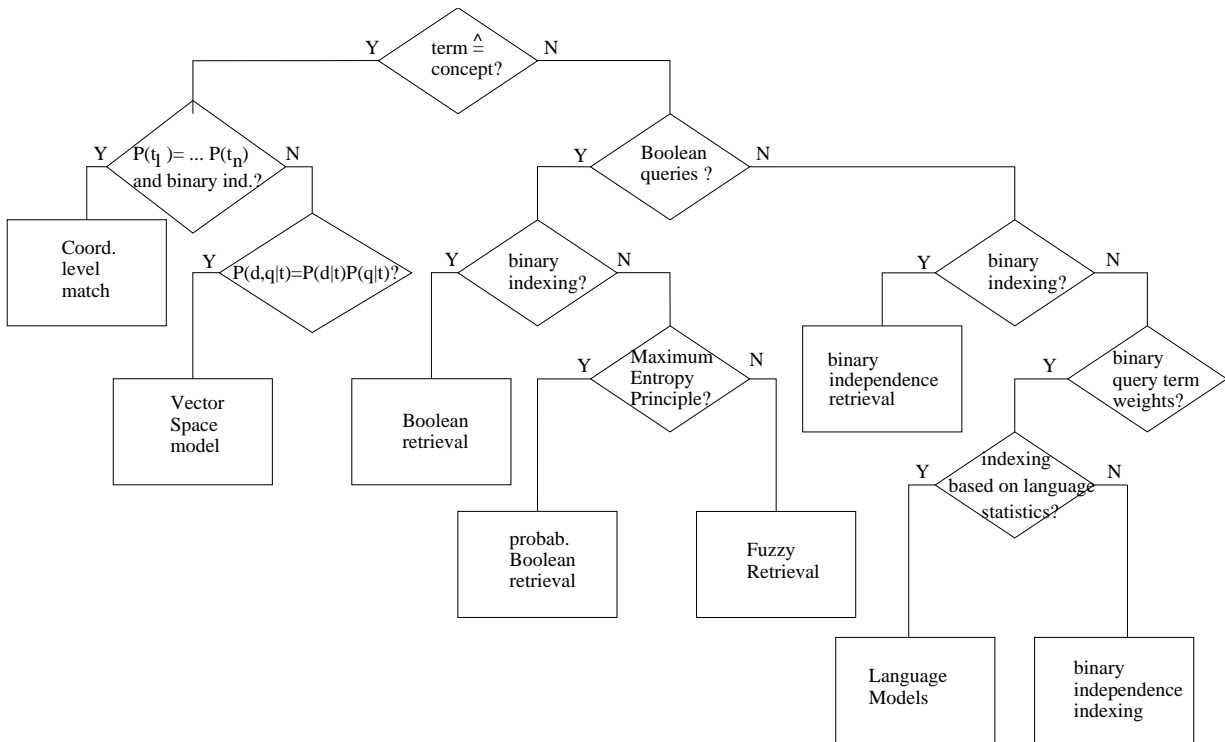


Figure 7.2: Classification of classical IR models

7.2.1 Disjoint basic concepts

Here we assume that terms $\{t_1 \dots t_n\}$ correspond to disjoint basic concepts: $t_i \cap t_j = \emptyset$ for $i \neq j$. Furthermore, let us assume that the terms for a complete cover of the concept space: $U = t_1 \cup t_2 \cup \dots \cup t_n$. So the terms form a dissection of U . This property can be used for computing the probability of the

implication $P(d \rightarrow q)$:

$$\begin{aligned}
 P(d \rightarrow q) &= P(q|d) \\
 &= \frac{P(d \cap q)}{P(d)} \\
 &= \frac{\sum_t P(d \cap q \cap t)}{P(d)} \\
 &= \frac{\sum_t P(d \cap q|t)P(t)}{P(d)}
 \end{aligned} \tag{7.1}$$

With the exception of the normalizing factor $P(d)$, the last equation defines the probability $P(d \rightarrow q)$ as sum of the probabilities of wrt. single terms $P(d \cap q|t)$. Each of these probabilities describes the relationship between the query q and the document d wrt. a single term t . In order to estimate these probabilities, we need additional assumptions.

As a straightforward approach, one can assume a uniform distribution over the set of terms, i.e. $P(t_1) = \dots = P(t_n)$. Treating documents and queries as sets of terms, we get a variant of the *coordination level match* where only the number of terms common to query and document is considered.

7.2.1.1 Vector space model

Now we show that a variant of the popular vector space model [Salton 71] can be explained in terms of our basic model. Here only the probabilities $P(d|t)$ and $P(q|t)$ are known. By applying the maximum entropy principle, we get the following independence assumption:

$$P(d \cap q|t) = P(d|t)P(q|t)$$

By combining this assumption with eqn (7.1), we get

$$\begin{aligned}
 P(d \rightarrow q) &= \frac{\sum_t P(d \cap q|t)P(t)}{P(d)} \\
 &= \frac{\sum_t P(d|t)P(q|t)P(t)}{P(d)} \\
 &= \sum_t P(d|t)P(q|t) \\
 &= \sum_t P(d \rightarrow t)P(t \rightarrow q)
 \end{aligned} \tag{7.2}$$

The two parameters in the last equation can be interpreted as follows:

$P(d \rightarrow t)$ describes the representation of a document d as the probability that document d implies term t . This kind of representation usually is called document indexing.

$P(t \rightarrow q)$ stands for the representation of a query q in terms of the probability that term t implies query q . These parameters often are called query indexing or query term weighting.

In order to show the analogy to the vector space model, we define document vectors $\mathbf{d} = (P(d \rightarrow t_1), \dots, P(d \rightarrow t_n))^T$ and query vectors $\mathbf{q} = (P(t_1 \rightarrow q), \dots, P(t_n \rightarrow q))^T$. Then eqn (7.2) can be rewritten as vector (dot) product:

$$P(d \rightarrow q) = \mathbf{d}^T \cdot \mathbf{q}$$

As an example, assume the following document vectors:

$$\begin{aligned}
 \mathbf{d}_1 &= (0, 1/3, 2/3) & \mathbf{d}_2 &= (1/3, 2/3, 0) \\
 \mathbf{d}_3 &= (1/2, 0, 1/2) & \mathbf{d}_4 &= (3/4, 1/4, 0)
 \end{aligned}$$

Given the query vector $\mathbf{q} = (1/5, 0, 2/3)^T$ we can compute the probability of implication for document d_1 as follows:

$$\begin{aligned}
 P(d_1 \rightarrow q_1) &= \sum_t P(d \rightarrow t)P(t \rightarrow q) = \mathbf{d} \cdot \mathbf{q} \\
 &= 0 \cdot \frac{1}{5} + \frac{1}{3} \cdot 0 + \frac{2}{3} \cdot \frac{2}{3} = \frac{4}{9}
 \end{aligned}$$

For the other documents in our example, we get $P(d_2 \rightarrow q_1) = 1/15$, $P(d_3 \rightarrow q_1) = 11/30$ and $P(d_4 \rightarrow q_1) = 3/20$. Thus, we arrive at the following ranking: $\{d_1\}, \{d_3\}, \{d_4\}, \{d_2\}$.

For practical applications, a number of heuristic weighting formulas has been developed for the vector space model as well as for related probabilistic models. According to the two major factors in these formulas, they are called $tf \times idf$ weights. Here we briefly describe a formula that is widely used at the moment.

First, we introduce a number of parameters:

- $T(d)$ set of terms occurring in d ,
- $l(d)$ length of document d ,
- al average length of a document in the collection,
- $df(t)$ document frequency of t (# docs containing t),
- $tf(t, d)$ within-document frequency of term t in document d ,
- N_d number of documents in the collection.

Now the inverse document frequency of term t wrt. a collection is defined as follows

$$idf(t) = \frac{\log \frac{N_d}{df(t)}}{N_d + 1}.$$

In addition, we need the normalized term frequency of term t wrt. document d :

$$ntf(t, d) = \frac{tf(t, d)}{tf(t, d) + 0.5 + 1.5 \frac{l(d)}{al}}$$

Then the document indexing weight of term t wrt. d is defined as

$$tfidf(t, d) = ntf(t, d) \cdot idf(t).$$

In order to fit into our model, an additional normalization would be required such that $\sum_{t \in d} tfidf(t, d) = 1$.

7.2.2 Nondisjoint basic concepts

Now we consider the case where terms represent nondisjoint concepts, i.e. there are terms t_i, t_j with $t_i \cap t_j \neq \emptyset$. However, we still assume that the terms form a complete cover of the concept space U .

In order to apply our framework model, we map terms onto disjoint atomic concepts in the following way: We form complete conjuncts (or minterms) of all terms t , in which each term occurs either positively or negated, i.e.

$$\begin{aligned} m_0 &= \bar{t}_1 \cap \bar{t}_2 \cap \bar{t}_3 \cap \cdots \bar{t}_{n-1} \cap \bar{t}_n \\ m_1 &= t_1 \cap \bar{t}_2 \cap \bar{t}_3 \cap \cdots \bar{t}_{n-1} \cap \bar{t}_n \\ m_2 &= \bar{t}_1 \cap t_2 \cap \bar{t}_3 \cap \cdots \bar{t}_{n-1} \cap \bar{t}_n \\ m_3 &= \bar{t}_1 \cap \bar{t}_2 \cap t_3 \cap \cdots \bar{t}_{n-1} \cap \bar{t}_n \\ &\vdots \\ m_{2^{n-2}} &= \bar{t}_1 \cap t_2 \cap t_3 \cap \cdots t_{n-1} \cap t_n \\ m_{2^{n-1}} &= t_1 \cap t_2 \cap t_3 \cap \cdots t_{n-1} \cap t_n \end{aligned}$$

Figure 7.3 illustrates this approach for the case of three terms. Based on this type of disjoint concepts, Boolean, fuzzy and probabilistic retrieval models can be explained.

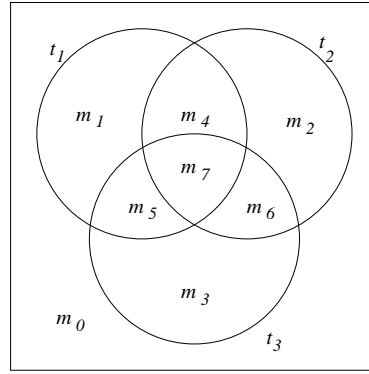


Figure 7.3: Construction of disjoint concepts for the case of three terms

7.2.2.1 Boolean retrieval

For Boolean retrieval, we assume binary indexing of documents, where each document is represented as a single atomic concept:

$$d = m_d = t_1^{\alpha_1} \cap \dots \cap t_n^{\alpha_n} \quad \text{with} \quad t_i^{\alpha_i} = \begin{cases} t_i & \text{if } \alpha_i = 1, \\ \bar{t}_i & \text{if } \alpha_i = 0. \end{cases}$$

Here assume a close world, that is, all terms not occurring within a document d are assumed to be negated, e.g.

$$\begin{aligned} d_1 &= \{t_1, t_3, t_4\} \\ &\hat{=} t_1 \cap \bar{t}_2 \cap t_3 \cap t_4 \cap \bar{t}_5 \cap \dots \cap \bar{t}_n \end{aligned}$$

By mapping terms onto disjoint concepts, we can represent terms as union of the corresponding basic concepts:

$$t_i = m_{i_1} \cup \dots \cup m_{i_r},$$

For example, term t_1 can be expressed as $t_1 = m_1 \cup m_4 \cup m_5 \cup m_7$ (see figure 7.3).

For a given Boolean query, we construct the corresponding disjunctive normal form, thus giving us a set of minterms. Thus, any query is mapped onto a set of minterms:

$$q = \bigcup m_{q_i}$$

Based on these assumptions, we can compute the probability of implication as follows:

$$\begin{aligned} P(d \rightarrow q) &= \frac{P(q \cap d)}{P(d)} \\ &= \frac{P(q \cap m_d)}{P(m_d)} \\ &= \begin{cases} 1 & \text{if } m_d \subseteq q, \\ 0 & \text{if } m_d \not\subseteq q. \end{cases} \end{aligned}$$

Boolean retrieval always yields a set of documents as result, without any further ranking; this feature is due to the fact that each document corresponds to a minterm, and a query is a set of minterms. From a theoretical point of view, a Boolean retrieval system only has to decide whether or not a document belongs to the minterms as specified by the query.

Let us consider an example with three terms, thus leading to eight minterms depicted in figure 7.3. For the (binary) document-term matrix shown in figure 7.4, we get the representation as minterms shown

	t_1	t_2	t_3	
d_1	0	1	1	$d_1 = m_6 = \bar{t}_1 \cap t_2 \cap t_3$
d_2	1	1	0	$d_2 = m_3 = t_1 \cap t_2 \cap \bar{t}_3$
d_3	1	0	1	$d_3 = m_5 = t_1 \cap \bar{t}_2 \cap t_3$
d_4	1	1	0	$d_4 = m_3 = t_1 \cap t_2 \cap \bar{t}_3$

Figure 7.4: Example: document representations for Boolean retrieval

in the same figure. The query

$$\begin{aligned}
 q_2 &= (t_1 \cup t_2) \cap t_3 \\
 &= (t_1 \cap t_2 \cap t_3) \cup (t_1 \cap \bar{t}_2 \cap t_3) \cup (\bar{t}_1 \cap t_2 \cap t_3) \\
 &= m_7 \cup m_5 \cup m_6
 \end{aligned}$$

leads to the answer set $\{d_1, d_3\}$, due to the fact that their minterms are contained within the query.

7.2.2.1.1 Fuzzy retrieval

Whereas Boolean retrieval is restricted to binary indexing of documents, fuzzy retrieval also can cope with weighted indexing (in the presence of Boolean queries). For single-term queries, we have

$$P(d \rightarrow q) = \frac{P(q \cap d)}{P(d)} = \frac{P(t \cap d)}{P(d)} = P(t|d) = P(d \rightarrow t)$$

When we have a Boolean combination of query terms, then there are different possibilities for computing the resulting weights. Following a probabilistic approach, one can assume the index weights to be independent of each other, thus leading to the following definitions:

$$\begin{aligned}
 P(d \rightarrow \bar{q}) &= \frac{P(\bar{q} \cap d)}{P(d)} = \frac{P(\bar{q} \cap d)}{P(d)} = 1 - P(q|d) = 1 - P(d \rightarrow q) \\
 P(d \rightarrow q \cap q') &= P(q \cap q'|d) \approx P(q|d)P(q'|d) = P(d \rightarrow q)P(d \rightarrow q') \\
 P(d \rightarrow q \cup q') &= P(q \cup q'|d) \approx P(q|d) + P(q'|d) - P(q|d)P(q'|d) \\
 &= P(d \rightarrow q) + P(d \rightarrow q') - P(d \rightarrow q)P(d \rightarrow q')
 \end{aligned}$$

Whereas this interpretation is based on the maximum entropy principle, the standard fuzzy interpretation is based on the principle of minimum entropy, thus leading to the following definitions for conjunction and disjunction:

$$\begin{aligned}
 P(d \rightarrow q \cap q') &= P(q \cap q'|d) \approx \min(P(q|d), P(q'|d)) = \min(P(d \rightarrow q), P(d \rightarrow q')) \\
 P(d \rightarrow q \cup q') &= P(q \cup q'|d) \approx \max(P(q|d), P(q'|d)) = \max(P(d \rightarrow q), P(d \rightarrow q'))
 \end{aligned}$$

7.2.2.2 Probabilistic retrieval

As the most important representative of a number of probabilistic IR models, we describe the binary independence retrieval (BIR) model [Robertson & Sparck Jones 76] here.

Like in Boolean retrieval, the BIR model is based on binary document indexing, thus representing a document as a single atomic concept:

$$d = m_d = t_1^{\alpha_1} \cap \dots \cap t_n^{\alpha_n}$$

Instead of the probability of implication $P(d \rightarrow q)$, we consider a monotone transformation of this parameter, namely the logg-odds transformation. Furthermore, we apply Bayes' theorem:

$$\begin{aligned} \log \frac{P(d \rightarrow q)}{1 - P(d \rightarrow q)} &= \log \frac{P(q|d)}{P(\bar{q}|d)} \\ &= \log \frac{P(d|q)}{P(d|\bar{q})} + \log \frac{P(q)}{P(\bar{q})} \\ &= \log \frac{P(t_1^{\alpha_1} \cap \dots \cap t_n^{\alpha_n} | q)}{P(t_1^{\alpha_1} \cap \dots \cap t_n^{\alpha_n} | \bar{q})} + \log \frac{P(q)}{P(\bar{q})} \end{aligned} \quad (7.3)$$

For the distribution of terms within relevant and nonrelevant documents, we assume linked dependence [Cooper 95]:

$$\frac{P(t_1^{\alpha_1} \cap \dots \cap t_n^{\alpha_n} | q)}{P(t_1^{\alpha_1} \cap \dots \cap t_n^{\alpha_n} | \bar{q})} = \frac{\prod_{i=1}^n P(t_i^{\alpha_i} | q)}{\prod_{i=1}^n P(t_i^{\alpha_i} | \bar{q})}$$

This assumption is less strict than the independence assumption mentioned in [Wong & Yao 95]. Combining the linked dependence with eqn (7.3), we get:

$$\begin{aligned} \log \frac{P(d \rightarrow q)}{1 - P(d \rightarrow q)} &= \log \frac{\prod_{i=1}^n P(t_i^{\alpha_i} | q)}{\prod_{i=1}^n P(t_i^{\alpha_i} | \bar{q})} + \log \frac{P(q)}{P(\bar{q})} \\ &= \sum_{i=1}^n \log \frac{P(t_i^{\alpha_i} | q)}{P(t_i^{\alpha_i} | \bar{q})} + \log \frac{P(q)}{P(\bar{q})} \end{aligned} \quad (7.4)$$

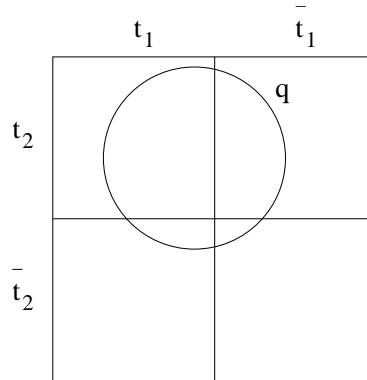


Figure 7.5: Example of BIR model with two terms

Figure 7.5 shows a simple example for the last formula. Here our vocabulary consists of two terms only, thus leading to four basic concepts which are represented as small squares in this figure. A document representation also corresponds to a basic concept, thus any document in the collection belongs to one of the four basic concepts here. In contrast, queries can be arbitrary subsets of the concept space.

In order to apply eqn 7.4, we have to estimate $P(t_i^{\alpha_i} | q)$ and $P(t_i^{\alpha_i} | \bar{q})$ for each term (in addition to $P(q)$ and $P(\bar{q})$). For example, with $\alpha_i = 1$, the probability $P(t_i | q)$ corresponds to the fraction of q that is covered by $t_i \cap q$ in the concept space; vice versa, for $\alpha_i = 0$ the probability $P(\bar{t}_i | \bar{q})$ denotes the ratio between $P(\bar{t}_i \cap \bar{q})$ and $P(\bar{q})$. Subsequently, we use the notations $u_i = P(t_i | q)$ and $v_i = P(t_i | \bar{q})$.

For $\alpha_i = 0$, the corresponding parameters can be computed as counter probabilities, i.e. $P(\bar{t}_i | q) = 1 - u_i$ and $P(\bar{t}_i | \bar{q}) = 1 - v_i$. Now we use a trick for expressing the probabilities $P(t_i^{\alpha_i} | q)$ (and analogously for \bar{q}) in a closed formula:

$$P(t_i^{\alpha_i} | q) = u_i^{\alpha_i} (1 - u_i)^{1 - \alpha_i} \quad \text{and} \quad P(t_i^{\alpha_i} | \bar{q}) = v_i^{\alpha_i} (1 - v_i)^{1 - \alpha_i}$$

By substituting these parameters in eqn 7.4, we get

$$\begin{aligned} \log \frac{P(d \rightarrow q)}{1 - P(d \rightarrow q)} &= \sum_{i=1}^n \log \frac{u_i^{\alpha_i} (1 - u_i)^{1 - \alpha_i}}{v_i^{\alpha_i} (1 - v_i)^{1 - \alpha_i}} + \log \frac{P(q)}{P(\bar{q})} \\ &= \sum_{i=1}^n \alpha_i \log \frac{u_i (1 - v_i)}{(1 - u_i) v_i} + \sum_{i=1}^n \log \frac{(1 - u_i)}{(1 - v_i)} + \log \frac{P(q)}{P(\bar{q})} \end{aligned} \quad (7.5)$$

In the last equation, only the first sum depends on the specific document, whereas the other addends are constant for a query. In most practical applications, one is only interested in the ranking of documents. Thus we only consider the first sum, for which we need the parameters u_i and v_i for all terms. In addition, one usually assumes that $u_i = v_i$ for all terms not included in the query formulation, thus restricting the evaluation of this sum to the query terms.

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}
BIR	0.76					0.69					0.48					0.40				
$P(d \rightarrow q)$	0.80					0.67					0.50					0.33				

Table 7.1: Example parameter estimation for the BIR model

Table 7.1 shows an example for the application of the BIR model. Here we have relevance judgements from 20 documents, from which we can estimate the following parameters:

$$\begin{aligned} u_1 &= P(t_1|q) = 8/12 & u_2 &= P(t_2|q) = 7/12 \\ v_1 &= P(t_1|\bar{q}) = 3/8 & v_2 &= P(t_2|\bar{q}) = 3/8 \end{aligned}$$

Substituting these estimates in eqn 7.5 (in addition, we have $P(q) = 12/20$ here), we get the values shown in the row titled ‘‘BIR’’ in table 7.1. These estimates can be compared with the values that could be derived directly for the four possible document representations in this example (row ‘‘ $P(d \rightarrow q)$ ’’). Obviously, the values in the two rows are different, but the ranking between the four classes of documents remains unchanged. The difference is due to the linked dependence assumption employed in the BIR model, which is only an approximation to reality.

The major advantage of the BIR model over a direct estimation of the probabilities $P(d \rightarrow q)$ does not become apparent in this example: When we have a larger number n of query terms, then the BIR model requires the estimation of $2n$ parameters. In contrast, we would have 2^n different representations, each requiring its own parameter. Furthermore, there is a big difference in the basis from which these parameters have to be derived: The BIR model subdivides the feedback set into relevant and nonrelevant documents only, from which the conditional probabilities have to be estimated for each term considered. In contrast, direct estimation would form 2^n disjoint subsets of the feedback set; thus, direct estimation is not applicable in practice.

7.2.2.3 The Probabilistic Indexing Model

The second probabilistic model we want to consider here is the binary independence indexing (BII [Fuhr & Buckley 91]), which 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 wrt. a number of documents, the BII model observes one document in relation to a number of queries submitted to the system. As a consequence, now a query is represented as a single atomic concept

$$q = m_q = t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n}$$

with

$$t_i^{\beta_i} = \begin{cases} t_i & \text{if } \beta_i = 1, \\ \bar{t}_i & \text{if } \beta_i = 0. \end{cases}$$

In addition, we consider the implication in the opposite direction ($q \rightarrow d$); like with the BIR model, we apply the log-odds transformation:

$$\begin{aligned} \log \frac{P(q \rightarrow d)}{1 - P(q \rightarrow d)} &= \log \frac{P(q|d)}{P(q|\bar{d})} + \log \frac{P(d)}{P(\bar{d})} \\ &= \log \frac{P(t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n} | d)}{P(t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n} | \bar{d})} \end{aligned} \quad (7.6)$$

$$+ \log \frac{P(d)}{P(\bar{d})} \quad (7.7)$$

Our linked dependence assumption in this case can be formulated as follows:

$$\frac{P(t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n} | d)}{P(t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n} | \bar{d})} = \frac{\prod_{i=1}^n P(t_i^{\beta_i} | d)}{\prod_{i=1}^n P(t_i^{\beta_i} | \bar{d})}$$

Combining this assumption with eqn 7.7, we get

$$\begin{aligned} \log \frac{P(q \rightarrow d)}{1 - P(q \rightarrow d)} &= \log \frac{P(t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n} | d)}{P(t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n} | \bar{d})} + \log \frac{P(d)}{P(\bar{d})} \\ &= \log \frac{\prod_{i=1}^n P(t_i^{\beta_i} | d)}{\prod_{i=1}^n P(t_i^{\beta_i} | \bar{d})} + \log \frac{P(d)}{P(\bar{d})} \\ &= \sum_{i=1}^n \log \frac{P(t_i^{\beta_i} | d)}{P(t_i^{\beta_i} | \bar{d})} + \log \frac{P(d)}{P(\bar{d})} \end{aligned} \quad (7.8)$$

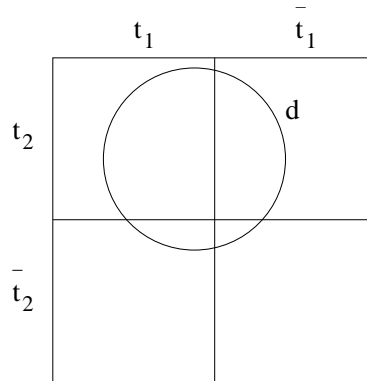


Figure 7.6: Example of BII model with two terms

Figure 7.6 shows a simple example for the last formula. Here our vocabulary consists of two terms only, thus leading to four basic concepts which are represented as small squares in this figure. A query representation also corresponds to a basic concept, thus any document in the collection belongs to one of the four basic concepts here. In contrast, documents can be arbitrary subsets of the concept space.

7.2.2.4 Language models

In the models discussed so far, the issue of document indexing has not been addressed; all these models assume that e.g. the probabilities $P(d|t)$ or $P(t|d)$ are given, without specifying the mapping from a

given document text onto these parameters. The BII model in combination with the description-oriented approach presented above may be a slight exception to that, but this approach only gives a framework for estimating the required probabilities.

During the past few years, a new class of probabilistic models has been developed which addresses the issue of document indexing: *Language models* are based on statistical models of natural language; they derive the parameters required for retrieval from the statistical properties of the document and the underlying collection.

Here we present one of these models, namely the model presented by Hiemstra [Hiemstra 98]. The basic assumption is similar to the probabilistic models presented before, in that terms are nondisjoint concepts. Like the BII model, we regard the probability of the implication $q \rightarrow d$:

$$P(q \rightarrow d) \approx \sum_m P(q \rightarrow m)P(m \rightarrow d) \quad (7.9)$$

Also like the BII model, a query is assumed to be a single atomic concept $q = m_q = t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n}$.

Thus, we get for the probability of implication:

$$\begin{aligned} P(q \rightarrow d) &\approx P(q \rightarrow m_q)P(m_q \rightarrow d) \\ &= P(m_q|q)P(d|m_q) \\ &= P(d|m_q) \end{aligned}$$

Applying Bayesian inversion leads to

$$P(d|m_q) = P(d) \frac{P(m_q|d)}{P(m_q)} \quad (7.10)$$

Next we assume independence of terms.

$$P(t_1^{\beta_1} \cap \dots \cap t_n^{\beta_n} | d) = \prod_{i=1}^n P(t_i^{\beta_i} | d) \quad (7.11)$$

In contrast to the probabilistic models discussed before, relevance of documents is not considered here; thus, this assumption seems to be stronger than the linked dependence assumptions employed for the BII and the BIR models.

Combining this assumption with eqn 7.10, we get

$$P(d|m_q) = P(d) \frac{\prod_{i=1}^n P(t_i^{\beta_i} | d)}{P(m_q)} \quad (7.12)$$

$$= C \cdot P(d) \cdot \prod_{i=1}^n P(t_i^{\beta_i} | d) \quad (7.13)$$

where $1/C = P(m_q) = \sum_{d'} P(d', m_q)$. As additional assumption, we assume that the relevance of a document is only affected by those terms of the document occurring in the query. Thus, we can restrict the product to the query terms:

$$P(d|q) \approx C \cdot P(d) \cdot \prod_{t_i \subseteq q} P(t_i | d) \quad (7.14)$$

Since C is constant for a given query, its value is not needed for computing a ranking wrt. a query. So only the parameters $P(d)$ and $P(t|d)$ have to be estimated. For $P(t|d)$, there is the problem of sparse data - especially for those terms not occurring within the document d . In order to solve this problem, this parameter is estimated from a mixture of the maximum likelihood estimates of $P(t)$ and $P(t|d)$; the former denotes the probability of the term occurring in a random document of the collection, whereas the latter is the probability for the specific document. As mixture formula, Hiemstra proposes a weighted sum:

$$\begin{aligned} P(t_i|d) &= \alpha_1 P(t_i) + \alpha_2 P(t_i|d) \\ &\text{with } 0 < \alpha_1, \alpha_2 < 1 \text{ and } \alpha_1 + \alpha_2 = 1 \end{aligned} \quad (7.15)$$

(The language model presented in [Ponte & Croft 98] proposes a risk function based on a geometric distribution for this purpose.) The estimation of these parameters is similar to the $tf \times idf$ weighting formula: Let

N_d number of documents in the collection,

$tf(t, d)$ within-document frequency of term t in document d ,

$df(t)$ document frequency of t (# docs containing t).

Then we can estimate

$$P(d) = \frac{1}{N_d} \quad (7.16)$$

$$P(t_i|d) = \alpha_1 \frac{df(t_i)}{\sum_t df(t)} + \alpha_2 \frac{tf(t_i, d)}{\sum_t tf(t, d)} \quad (7.17)$$

Kapitel 8

Models based on predicate logic

8.1 Introduction

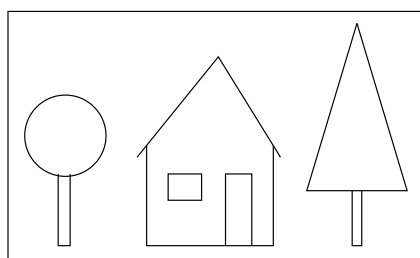


Abbildung 8.1: Example image

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 8.1, 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 ≠ 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.

8.2 Terminological logic

8.2.1 Thesauri

If we look at classical thesauri, then we see that a structure like e.g. in picture 8.2 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 ⇔ quadrangle ∧ regular-polygon.
```

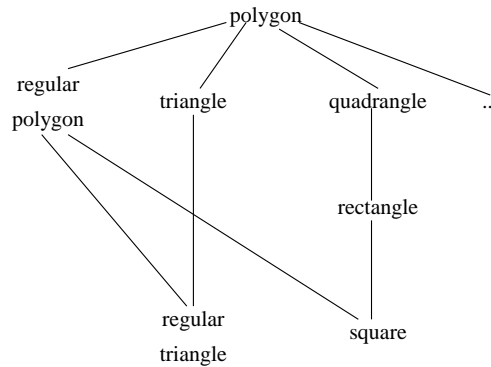


Abbildung 8.2: Thesaurus structure

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.

8.2.2 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 <· **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** <· **C** (or **D** <· **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:

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

For explaining the meaning of these constructs, let the symbols C, C_1, C_2, \dots stand for concepts and R, R_1, R_2, \dots for roles.

(and $C_1 C_2 \dots C_n$) denotes the set of all individuals that belong at the same time to concept C_1 and C_2 and $\dots C_n$. 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:

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

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

$$\begin{aligned}
 \text{student} & = \text{(and person (atleast 1 enrolled)} \\
 & \quad \text{(atmost 1 enrolled)} \\
 & \quad \text{(all enrolled university))} \\
 & = \text{(and person (exactly 1 enrolled)} \\
 & \quad \text{(all enrolled university))} \\
 & = \text{(and person (func enrolled university))}
 \end{aligned}$$

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

8.2.3 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

- $i_1 \neq i_2 \implies \mathcal{I}(i_1) \neq \mathcal{I}(i_2)$,
- 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:

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

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

$$\begin{aligned} C[i] &\text{ iff } \mathcal{I}(i) \in \mathcal{I}(C) \\ R[i_1, i_2] &\text{ iff } \langle \mathcal{I}(i_1), \mathcal{I}(i_2) \rangle \in \mathcal{I}(R) \end{aligned}$$

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 .

8.2.4 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

8.2.4.1 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 example, 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)))

```

8.2.4.2 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 8.3.5).

8.3 Datalog

8.3.1 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`.

8.3.2 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 8.3), the cyclic link structure also raises the problem of possible cycles in the inference process; however, Datalog can cope with these cycles.

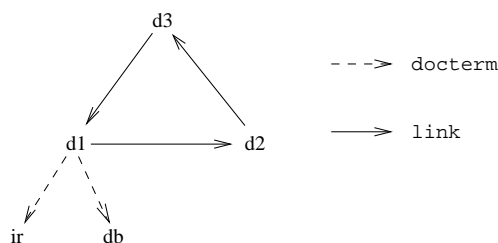


Abbildung 8.3: Hypertext retrieval

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

8.3.3 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 reasonable 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

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

(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.

8.3.4 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 8.4. 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.

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).
```

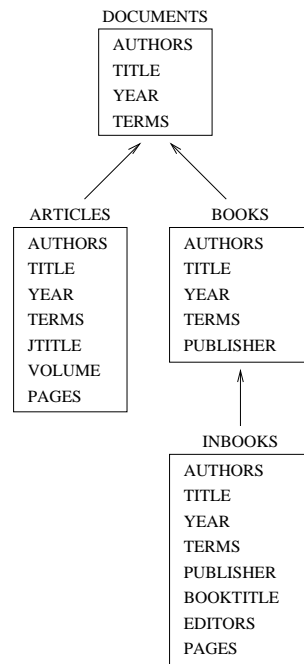


Abbildung 8.4: Example document class hierarchy

```

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 books 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).
  
```

8.3.5 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 8.2.4.2, 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-papers` 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.

8.4 Probabilistic Datalog

8.4.1 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 8.3, 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.

8.4.2 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 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, whereas 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.

8.4.3 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* $p(t_1, \dots, t_n)$ consists of an n -ary predicate symbol p and a list of arguments (t_1, \dots, t_n) such that each t_i is a term. A *literal* is an atom $p(t_1, \dots, t_n)$ or a negated atom $\neg p(t_1, \dots, t_n)$.

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 $\alpha = \alpha'$.

8.4.4 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 Datalog 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 $L_0: -L_1 \& \dots \& L_n$.

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. $(M, w, v) \models L_0: -L_1 \& \dots \& L_n$ iff whenever $\{L_1 v\} \in w \wedge \dots \wedge \{L_n v\} \in w$ 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 example, the program

0.9 docTerm(d1,ir).

has the interpretation

$$P(W_1) = 0.9: \{\text{docTerm}(d1,ir)\}$$

$$P(W_2) = 0.1: \{\}$$

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 :

$$P(W_1) = 0.2: \{\text{docTerm}(d1,ir)\}$$

$$P(W_2) = 0.2: \{\text{docTerm}(d1,ir), \text{docTerm}(d1,db)\}$$

$$P(W_3) = 0.3: \{\text{docTerm}(d1,db)\}$$

$$P(W_4) = 0.3: \{\}$$

I_2 :

$$P(W_1) = 0.5: \{\}$$

$$P(W_2) = 0.4: \{\text{docTerm}(d1,ir), \text{docTerm}(d1,db)\}$$

$$P(W_3) = 0.1: \{\text{docTerm}(d1,db)\}$$

I_3 :

$$P(W_1) = 0.4: \{\text{docTerm}(d1,ir)\}$$

$P(W_2) = 0.5: \{\text{docTerm}(d1, db)\}$

$P(W_3) = 0.1: \{\}$

Thus, we get for the probability of the answer:

$0.0 \leq P(\text{docTerm}(d1, ir) \& \text{docTerm}(d1, db)) \leq 0.4,$

depending on the overlap between the worlds in which $\text{indterm}(d1, ir)$ is true and those in which $\text{indterm}(d1, db)$ 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.

$0.3 \text{ indterm}(d1, ir) \ \& \ \text{indterm}(d1, db).$

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

$P(\text{docTerm}(d1, ir) \& \text{docTerm}(d1, db)) = 0.2.$

The general procedure is described in more detail below.

8.4.5 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 \mathbf{EK} of event keys which is a set of unique identifiers plus two special elements \top and \perp for events (facts) which are always true or false, respectively. Now we define a mapping $\varepsilon : \bar{P}_E \rightarrow \mathbf{EK} - \{\perp\}$ 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. $\eta(L_0: -L_1 \& \dots \& L_n v) := \eta(L_1 v) \wedge \dots \wedge \eta(L_n v)$
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 \mathbf{EE} denote the set of all event expressions that can be formed this way. Then \mathbf{EE} together with the Boolean operators forms a Boolean algebra.

Given the set \mathbf{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 \mathbf{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 \mathbf{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:

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

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:

$$\begin{aligned} P(e) &= P(K_1 \vee \dots \vee K_n) \\ &= \sum_{i=1}^n (-1)^{i-1} \left(\sum_{\substack{1 \leq j_1 < \dots < j_i \leq n}} P(K_{j_1} \wedge \dots \wedge K_{j_i}) \right). \end{aligned} \quad (8.1)$$

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:

$$\begin{aligned} &P(i(d1,ir) \ \& \ \neg i(d1,db) \ | \ \neg i(d1,ir) \ \& \ i(d1,db)) = \\ &P(i(d1,ir) \ \& \ \neg i(d1,db)) + P(\neg i(d1,ir) \ \& \ i(d1,db)) - \\ &\quad P(i(d1,ir) \ \& \ \neg i(d1,db) \ \& \ \neg i(d1,ir) \ \& \ i(d1,db)) = \\ &P(i(d1,ir) \ \& \ \neg i(d1,db)) + P(\neg i(d1,ir) \ \& \ i(d1,db)) \end{aligned}$$

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.

8.4.6 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 efficiently. 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 8.1, 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:

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

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.

$$0.2 \text{ py}(d3,89). \quad 0.7 \text{ py}(d3,90). \quad 0.1 \text{ py}(d3,91).$$

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:

$$P(W_1) = 0.2: \{\text{py}(d3,89)\}$$

$$P(W_2) = 0.7: \{\text{py}(d3,90)\}$$

$$P(W_3) = 0.1: \{\text{py}(d3,91)\}$$

Now a query for books published after 89

$$?- \text{py}(X,Y) \ \& \ Y > 89.$$

would yield the event expression $[p(d3,90) \mid p(d3,91)]$ for *d3*. In terms of the sieve formula, we have $P(p(d3,90) \mid p(d3,91)) = P(p(d3,90)) + P(p(d3,91)) - P(p(d3,90) \ \& \ p(d3,91))$.

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 seen 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

$$P(e_1 \wedge e_2) = 0.$$

8.4.7 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:

$$0.8 \text{ docTerm}(d1,db). \quad 0.7 \text{ docTerm}(d1,ir).$$

$$0.4 \text{ qtw}(db). \quad 0.6 \text{ qtw}(ir).$$

$$s(D) \text{ :- } \text{qtw}(X) \ \& \ \text{docTerm}(D,X).$$

$$?- \text{ s}(D).$$

The interpretation of this program depicted in figure 8.5 shows that the worlds of $\text{qtw}(db)$ and $\text{qtw}(ir)$ are disjoint, whereas $\text{docTerm}(d1,db)$ and $\text{docTerm}(d1,ir)$ are independent events. As event expression of the result, we get

$$d1 \ [q(db) \ \& \ dT(d1,db) \ \mid \ q(ir) \ \& \ dT(d1,ir)].$$

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, disk size and price):

$$\text{pc}(m1,486/dx50,8,540,900).$$

$$\text{pc}(m2,pe60,16,250,1000).$$

$$\text{pc}(m3,pe90,16,540,1100).$$

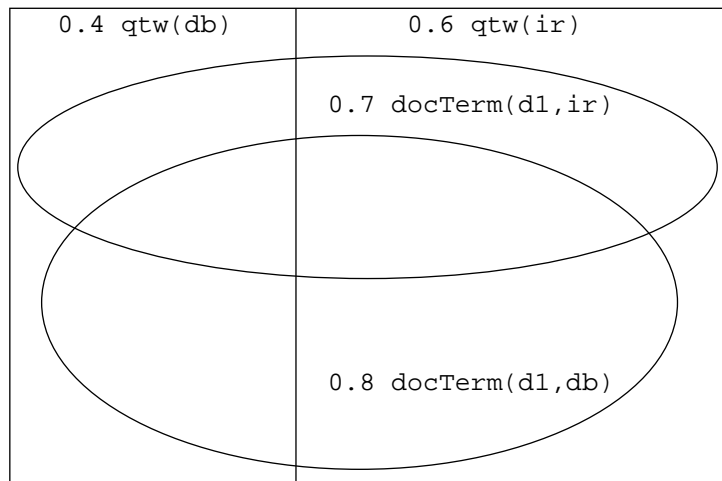


Abbildung 8.5: Interpretation of query term weighting example

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  $\hat{<}$ (1000,900)
1.00  $\hat{<}$ (1000,950)
0.99  $\hat{<}$ (1000,1000)
0.90  $\hat{<}$ (1000,1050)
0.60  $\hat{<}$ (1000,1100)
```

Thus, formulating the query using this predicate

```
?- pc(MOD, CPU, MEM, DISK, PRICE), PRICE  $\hat{<}$  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.

8.4.8 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:

$P(W_1) = 0.7: \{\text{man}(\text{peter}), \text{likes-sports}(\text{peter})\}$
 $P(W_2) = 0.3: \{\text{man}(\text{peter})\}$

In terms of event expressions, the query

?- likes-sports(peter)

would generate the expression

$1-s_1(\text{peter}) \ \& \ \text{man}(\text{peter})$, where $1-s_1(\text{peter})$ is an event generated by the first rule, with a probability of 0.7.

As a similar example, consider the case where also the sex of a person is unknown:

```
0.7 1-s(X)      :- sex(X,male).
0.4 1-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 $\text{sex}(X,\text{male})$ and $\text{sex}(X,\text{female})$ are disjoint for any X , the interpretation of this program would be:

$P(W_1) = 0.35: \{\text{sex}(\text{peter},\text{male}), 1-s(\text{peter})\}$
 $P(W_2) = 0.15: \{\text{sex}(\text{peter},\text{male})\}$
 $P(W_3) = 0.20: \{\text{sex}(\text{peter},\text{female}), 1-s(\text{peter})\}$
 $P(W_4) = 0.30: \{\text{sex}(\text{peter},\text{female})\}$

Asking whether or not peter likes sports would yield the event expression

$(\text{sex}_1(\text{peter},\text{male}) \ \& \ 1-s_1(\text{peter}) \ | \ \text{sex}_2(\text{peter},\text{female}) \ \& \ 1-s_2(\text{peter})) \ \& \ \text{human}(\text{peter})$
 from which we can derive the correct probability $0.5 \cdot 0.7 + 0.5 \cdot 0.4 = 0.55$.

However, rules must be formulated 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).
```

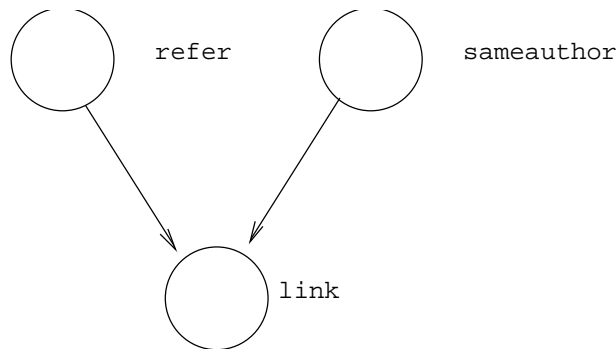


Abbildung 8.6: Example inference network

But what happens if there is both a reference link and the same author?

?? $\text{link}(D1,D2) \ :- \ \text{refer}(D1,D2) \ \& \ \text{sameauthor}(D1,D2)$.

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 $P(l|r \ \& \ s)$.

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 8.6). In probabilistic Datalog, the link matrix is specified by means of a set of rules

Kapitel 9

IR-Systeme

9.1 Ebenenarchitektur

Wie generell bei Informationssystemen, kann man auch bei IR-Systemen eine Mehrebenenarchitektur zugrunde legen. Diese ist in Abbildung 9.1 dargestellt.

Funktionalität	Konzeptionelle Ebene
	Präsentationsebene
	semantische Ebene
	logikorientierte Ebene
	physische Ebene

Abbildung 9.1: Ebenenarchitektur von Informationssystemen

- Die oberste Ebene ist die konzeptionelle Ebene. Hier wird die generelle Funktionalität aus Nutzersicht beschrieben.
- Die Präsentationsebene beschreibt die Benutzungsoberfläche des Systems, also die Visualisierung und die hier angebotenen Funktionen des Systems.
- Die semantische Ebene beschreibt die Objekte (einschließlich der darauf anwendbaren Funktionen), die vom Informationssystem verwaltet werden.
- Die logikorientierte Ebene beschreibt die unterste implementierungsunabhängige Schicht des Systems. Hier finden sich vor allem die Retrievalmodelle wieder, die in diesem Skript behandelt worden sind.
- Die physische Ebene behandelt schließlich die Zugriffsstrukturen und darauf operierenden Algorithmen des Systems.

Im folgenden beschreiben wir zunächst einige Ansätze zur Modellierung der konzeptionellen und der semantischen Ebene. Bezüglich der Präsentationsebene sei auf die ausgezeichnete Darstellung von Marti Hearst in [Baeza-Yates & Ribeiro-Neto 99] verwiesen. Die physische Ebene wird im nächsten Kapitel behandelt.

9.2 Konzeptionelle Ebene

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.

9.2.1 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: <ul style="list-style-type: none"> a) Only when searcher asks for suggestions. b) Always when it identifies a need.
4	Executes automatically. System executes actions automatically and then: <ul style="list-style-type: none"> a) Informs the searcher. b) Does not inform the searcher.

Table 9.1: Levels of system involvement

In Tabelle 9.1 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 Hilfsfunktionen 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.

9.2.2 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 multiple 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 9.2: Levels of search activities

Bezogen auf den Suchprozeß unterscheidet Bates die in Tabelle 9.2 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.

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 9.3: Selected example search tactics

Die in den Tabellen 9.3 und 9.4 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 9.5 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.*)

9.2.3 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 9.6 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 Forschung auf die Systembeteiligung der Stufen 1–3 für Taktiken und Stratageme konzentrieren, während

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 9.4: 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 (online or offline) and locates all references on one's topic of interest.

Table 9.5: Example stratagems

die Stufe 4 und / oder Strategien derzeit noch zu anspruchsvolle Aufgaben seien, um sie erfolgversprechend angehen zu können.

9.3 Semantic level

9.3.1 The FERMI multimedia retrieval model

There are three different views for multimedia documents, namely the logical, layout and content view (see e.g. [Meghini et al. 91]). The logical view deals with the logical structure of the document (e.g. chapters, sections and paragraphs) and the layout view with the presentation of the document on an output medium (e.g. pages and rectangular areas within a page). The content view addresses the semantic content of a document, and thus is the focus of IR.

The FERMI multimedia model (FMM) presented in [Chiaramella et al. 96] considers a content structure which is closely related to the logical structure (in contrast to classical IR models treating documents as atomic units). Thus, the answer to an IR query may not only return whole documents, but also substructures according to the logical structure. For this purpose, the FMM uses a representation for the logical structure which focuses on those elements which are important for retrieval, thus neglecting issues dealt with by other models (e.g. ODA, SGML) which also relate to other tasks (e.g. authoring, presentation).

A database is a set of documents, and each document is a tree consisting of typed structural objects (e.g. book, chapter, section, paragraph, image), where the leaves contain single-media data. Hypermedia documents also contain links between different nodes (possibly from different documents). Nodes are assigned attributes, which can be either standard attributes (e.g. author or creation date) or so-called index expressions describing the content of a node. The latter are initially assigned to the leaf nodes only, where the indexing language depends on the media type. For example, for text, there are languages

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 or wrong terms	NEIGHBOR TRACE SUPER SUB RELATE
Revise terms	SPACE RESPELL FIX REVERSE CONTRARY SUPER SUB RELATE
Revise search formulation	SPECIFY EXHAUST REDUCE PARALLEL PINPOINT BLOCK

Table 9.6: Tactics suggested in response to searcher request

for describing the physical, the structural and the symbolic content, whereas for images, there is in addition also a spatial and a perceptive content.

Depending on the class of an attribute, attribute values may be ascending or descending along the hierarchy. For example, the authors of different nodes are propagated upwards (involving an attribute-specific merge operation), whereas the publishing date of a complete document is propagated downwards. The index expressions assigned to leave nodes are also propagated upwards. Like any data model, the FMM also supports typing of nodes, links and attributes.

Retrieval in this model follows the logical approach, i.e. search for document nodes n which imply the query q . In order to consider the hierarchical structure of documents, the overall retrieval strategy is to search for the smallest possible units (i.e. low-level nodes) fulfilling this criterion. For example, assume a node n comprising two subnodes n_1 and n_2 , where n_1 is indexed with t_1 and n_2 with t_2 ; then n would be indexed with both t_1 and t_2 , and a query asking for these two terms would retrieve node n , but none of n_1 and n_2 . On the other hand, if there were a supernode n' , having n as subnode, then n' also would be indexed with t_1 and t_2 (plus possibly other terms from its other subnodes), but it would not be retrieved in response to $q = t_1 \wedge t_2$, since there is a subnode of n' (namely n) which also implies the query.

This strategy is supported by the upward propagation of index expressions. The original formulation of the FMM uses predicate logic as basic retrieval logic, whereas in [Lalmas 97], a refinement of the retrieval model (but restricted to propositional logic) using Dempster-Shafer theory is described. However, this work is based on propositional logic, thus leaving a huge gap between the semantic data model and the logic used for its implementation.

9.3.2 POOL: A probabilistic object-oriented logic

The motivation behind the development of POOL ([Rölleke 98]) was the need for a logic for retrieval of structured objects, like e.g. hypermedia documents. Its major features are the support of nested objects and the combination of a restricted form of predicate logic with probabilistic inference.

Objects in POOL have an identifier and a content, which is a POOL program. Objects with a nonempty content are also called *contexts*, because the logical formulas forming the content first are valid only within this object/context. Propagation of this knowledge into other contexts is performed via the process of augmentation (see below). A program is a set of clauses, where each clause may be either a context, a proposition or a rule. In the following example, we have several nested contexts: an article **a1** consisting of two sections **s11** and **s12**, where the latter again contains two subsections **ss121**, **ss122**. A *proposition* is either a *term*¹ (like **image**), a *classification* (e.g. **article(a1)**) or an *attribute* (e.g. **s11.author(smith)**). Propositions also may be negated (e.g. **not presentation**) or assigned a probability (e.g. **0.6 retrieval**).

```
a1[ s11[ image 0.6 retrieval presentation ]
    s12[ ss121[ audio indexing ]
        ss122[ video not presentation ] ] ]
s11.author(smith)
s121.author(miller) s122.author(jones)
a1.pubyear(1997)
article(a1) section(s11) section(s12)
  subsection(ss121) subsection(ss122)
docnode(D) :- article(D)
docnode(D) :- section(D)
docnode(D) :- subsection(D)
mm-ir-doc(D) :- docnode(D) &
                D[audio & retrieval]
german-paper(D) :- D.author.country(germany)
```

A *rule* consists of a head and a body, where the head is either a proposition or a context containing a proposition; the rule body is a conjunction of subgoals, which are propositions or contexts containing a rule body. In the example program shown above, the first three rules state that articles, sections and subsections are document nodes. Next, we classify documents talking both about **audio** and **retrieval**

¹Throughout this paper, we use the word “term” in the typical IR meaning, not in the usual logical meaning. Logically, terms stand for argument-free predicates.

as `mm-ir-doc`. We also allow for path expressions in rule bodies as shown in the last rule, which is a shorthand for `D.author(X) & X.country(germany)`.

A *query* consists of a rule body only, e.g. `?- D[audio & indexing]`, for which `ss121` would be an answer.

A basic assumption underlying POOL is that clauses only hold for the context in which they are stated. For example, the query `?- D[audio & video]` cannot be answered by an atomic context, since `audio` occurs in `ss121` and `video` in `ss122`. For dealing with content-based retrieval of aggregated contexts, POOL uses the concept of *augmentation*. For this purpose, propositions are propagated to surrounding contexts — similar to upward propagation of attribute values in the FMM. This way, the content of a context is augmented by the content of its components. Thus, the last query would be fulfilled by the context `s12`. However, augmentation also may lead to inconsistencies: when we ask `?- D[image & video]` and combine contexts `s11` and `s12`, then we get a contradiction with respect to `presentation`. In classical logic, this inconsistency would allow us to infer anything. In order to avoid these problems, POOL is based on four-valued logic (as described in [Rölleke & Fuhr 96]), treating only `presentation` as inconsistent and yielding `a1` as correct answer to the last query.

Since we have no inference engine which implements POOL directly, we map POOL programs onto probabilistic datalog programs, for which we use the HySpirit inference engine.

9.3.3 FMM and POOL

From the description of POOL given above, it is obvious that it is more general than the FMM. The FMM poses a number of reasonable restrictions on hypermedia documents (e.g. that only leaf nodes have a content, or the type hierarchy on document nodes) which are not present in POOL. On the other hand, both approaches deal with nested objects, and the retrieval strategy of the FMM is already integrated in POOL.

The only feature of FMM that we have to model explicitly in POOL is propagation of attribute values. This can be achieved by formulating an appropriate rule for each attribute, depending on the propagation direction, e.g.

```
D.author(A) :- D[S] & docnode(D) & docnode(S) &
              S.author(A)
S.pubyear(Y) :- D[S] & docnode(D) & docnode(S) &
               D.pubyear(Y)
```

The first rule performs (recursively) upward propagation of author names: When a document node `D` contains a subnode `S`, then any author of `S` also is an author of `D`. In a similar way, the second rule yields downward propagation of the publication year.

The consideration of hypertext links in retrieval is not an explicit part of the FMM. However, intuitively, when there is a link from node n_1 to n_2 , then the content of n_2 should be considered as being contained in n_1 in a similar way as that of any subnode of n_1 . Thus, a simple strategy for achieving this kind of behaviour in POOL would be the formulation of a rule like `D[P] :- D.link(C) & C[P]`, i.e. any proposition occurring in an object that is linked to object `D` is considered as being contained in `D`. However, our general idea behind POOL is such that the content-oriented retrieval strategy should not be formulated explicitly, so hypermedia retrieval should be a part of the implicit retrieval strategy of POOL.

Kapitel 10

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/XML) und von Zugriffspfadstrukturen, die in IRS eingesetzt werden (Scanning, invertierte Listen, Signaturen, PAT-Bäume).

10.1 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.

10.1.1 Speichermedien

Für IRS sind insbesondere optische Speichermedien relevant, weil sie hohe Speicherkapazitäten bei niedrigen Kosten bieten. Zur Zeit sind drei verschiedene Typen von optischen Platten auf dem Markt:

- **CD-ROM** besitzen eine Kapazität von 650 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. Besonders populär sind derzeit die einmal beschreibbaren CD-R, die die Herstellung von CDs in kleinen Stückzahlen ermöglichen.

Eine Weiterentwicklung der CD-ROM ist die **Digital Versatile Disk**. Die DVD kann ein- oder zweiseitig jeweils in einer oder in zwei Lagen beschrieben sein, woraus sich eine Kapazität zwischen 4,7 und 17 GB ergibt. Die Übertragungsrate beträgt mindestens 1.1 MB/s.

- **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. Am gängigsten ist das CD-Format mit ca. 650 MB (CD-R), das kaum teurer als CD-ROMs ist und daher sowohl zur Datensicherung als auch zur Verbreitung von Datenbeständen in kleinen Stückzahlen eingesetzt wird.

Für alle Arten von optischen Platten gibt es sogenannte Jukeboxen, die für eine einzelne Aufnahme-/Wiedergabestation mit Hilfe eines Wechslers mehrere Platten aufnehmen können.

10.1.2 Ein-/Ausgabegeräte

Zur maschinellen Erfassung von Dokumenten werden heute bereits in weitem Maße **Scanner** eingesetzt. Typische optische Auflösungen bewegen sich im Bereich von 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.

10.1.3 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. Die typische Übertragungsrates beträgt derzeit **100 Mbit/sec**, Backbones werden im Gigabit-Bereich betrieben.
- **ATM**-Netze erreichen Übertragungsrates 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.
- **ADSL** (Asymmetric Digital Subscriber Line) als Weiterentwicklung von ISDN erlaubt die Übertragung von bis zu 8 MBit/sec beim Empfang und bis zu 768 KBit/sec beim Senden. Aufgrund der Asymmetrie ist ADSL primär für typische Consumer-Anwendungen geeignet (Web-Browsen, Video on Demand).

10.2 Aufbau von IRS

10.2.1 Funktionale Sicht

Abbildung 10.1 (aus [Frakes & Baeza-Yates 92]) zeigt die funktionale Sicht auf ein IRS.

10.2.2 Dateistruktur

Abbildung 10.2 zeigt die Dateistruktur eines typischen IRS. Neben der **Dokument-Datei** mit den eigentlichen Dokumenten sind meist noch zwei zusätzliche Dateien vorhanden: Das **Wörterbuch** enthält die in den Dokumenten vorkommenden Terme zusammen mit ihrer Dokumentenhäufigkeit; 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.

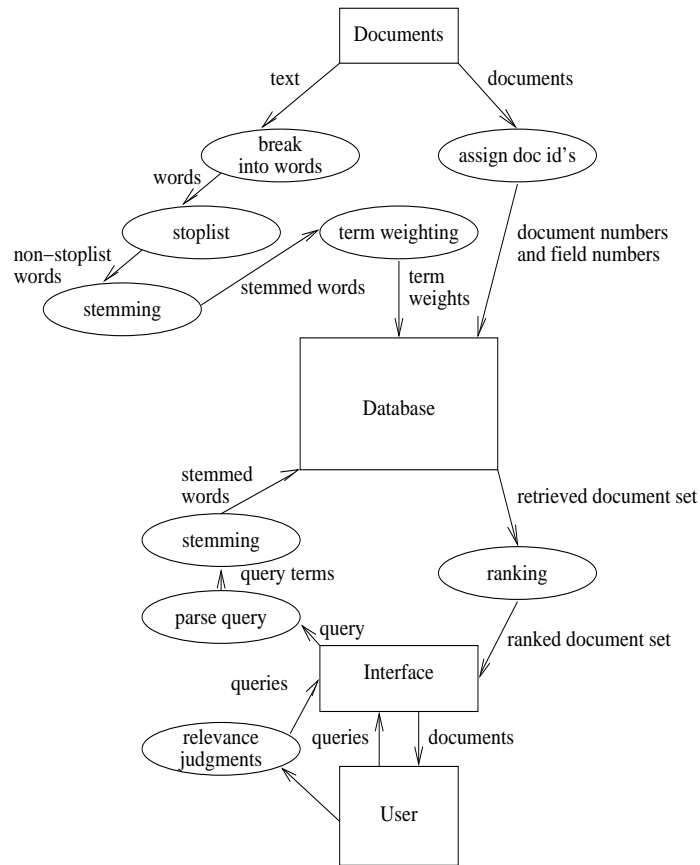


Abbildung 10.1: Funktionale Sicht eines IRS

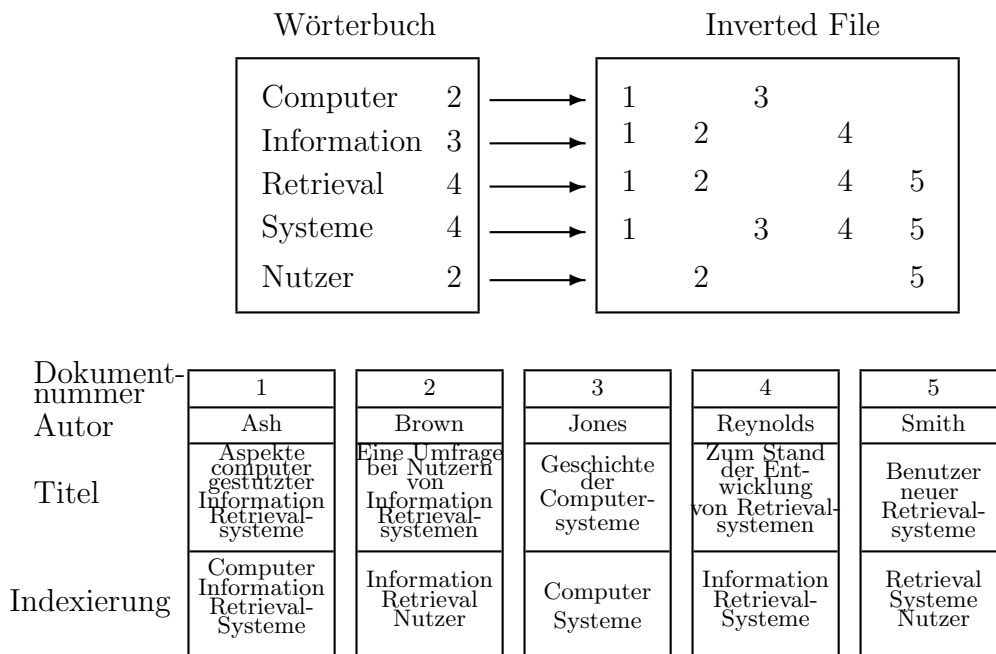


Abbildung 10.2: Dateistruktur eines IRS

10.2.3 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))

10.3 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- oder XML-Format verwalten).

In den folgenden Abschnitten stellen wir zwei Arten von standardisierten Dokumentformate vor: ODA ist eine primär europäische Entwicklung für den Bürobereich, die primär wegen der darin enthaltenen Konzepte interessant ist. Markup-Sprachen erführen zunächst durch das aus den USA stammende SGML eine gewisse Verbreitung, während der Nachfolgers XML heute durch das WWW in aller Munde ist.

10.3.1 ODA

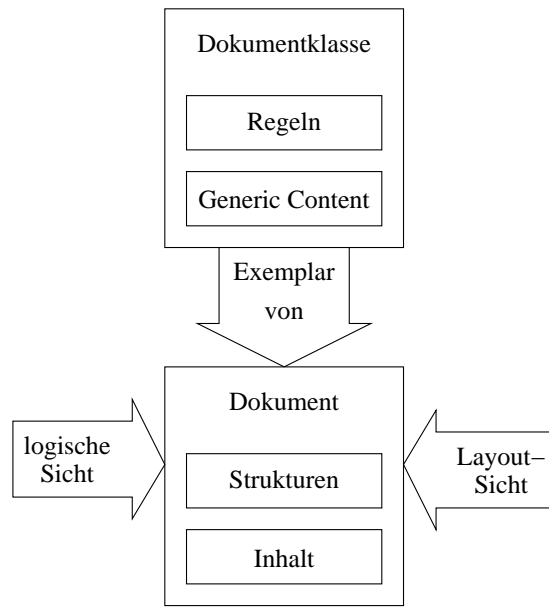
ODA (**Office Document Architecture**) definiert ein Dokumentarchitektur-Modell, das die Be- und Verarbeitung von Dokumenten durch unterschiedliche Systeme ermöglicht. Das zugehörige Austauschformat zur Übertragung von Dokumenten zwischen verschiedenen Systemen heißt ODIF (**Office Document Interchange Format**). Die nachfolgende Beschreibung orientiert sich im wesentlichen an der Darstellung in [Krönert 88].

10.3.1.1 Grundkonzepte 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).



Dokumente aus der Sicht von ODA

Abbildung 10.3: Dokumente aus der Sicht von ODA

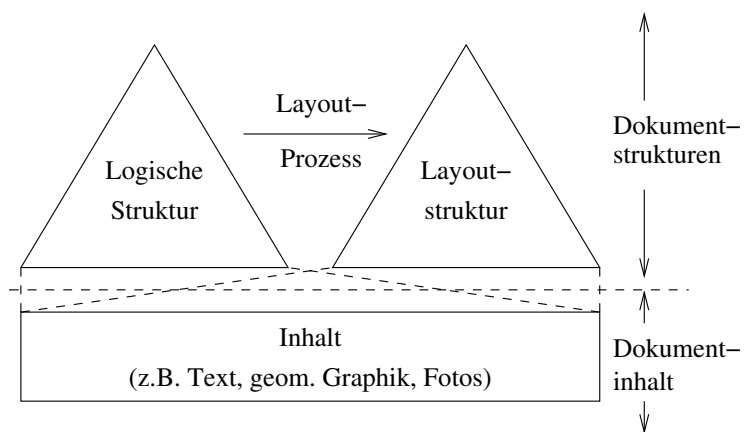


Abbildung 10.4: Dokumentstrukturen und Dokumentinhalt

Der eigentliche Inhalt¹ 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.

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 Standardparagrafen.

10.3.1.2 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.

10.3.1.3 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:

- 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),

¹hier nicht im Sinne der inhaltlichen Bedeutung eines Textes

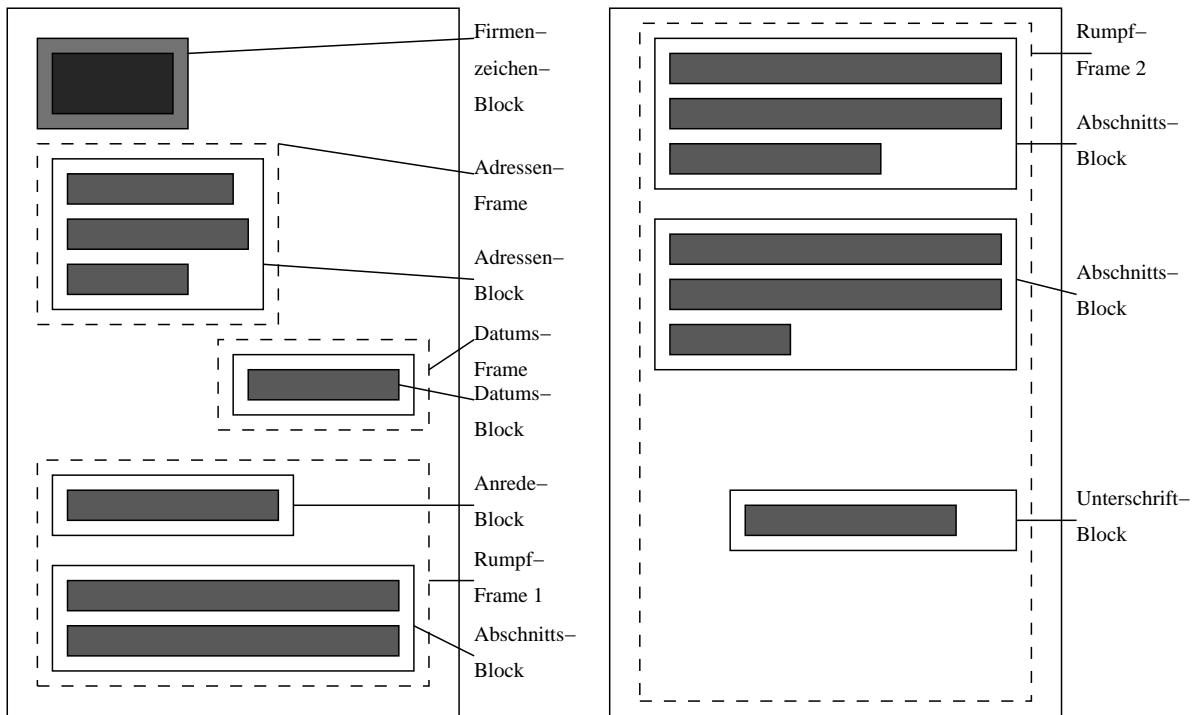


Abbildung 10.5: Aufteilung eines Dokumentes in Blöcke und Frames

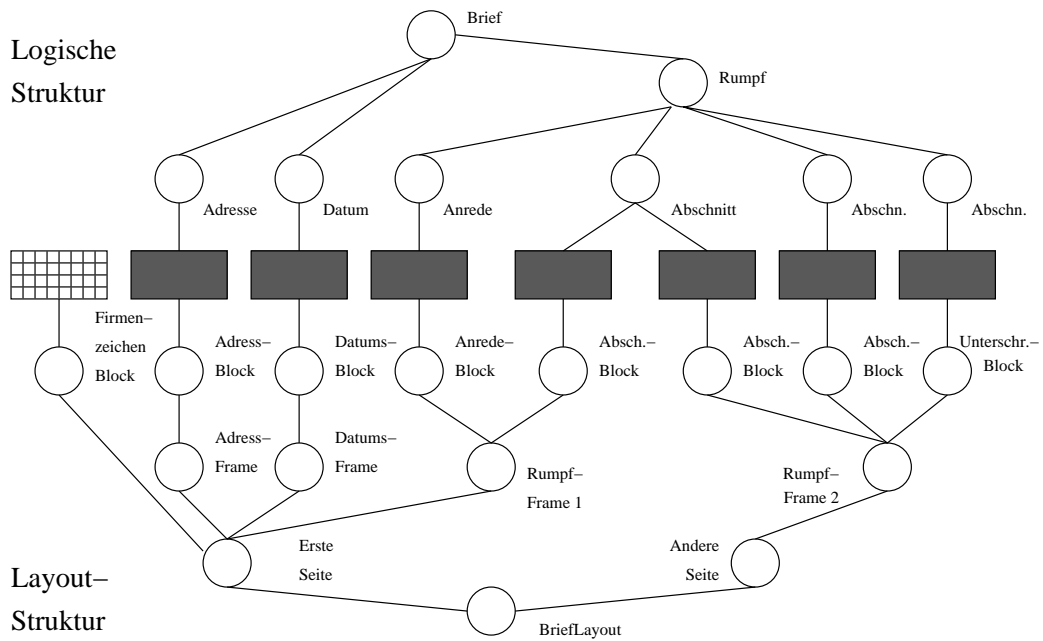


Abbildung 10.6: Logische und Layout-Sicht desselben Dokumentes

- 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.

10.3.2 Markup-Sprachen

10.3.2.1 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. Satzende- und Abkürzungspunkte unterscheiden zu können (in \LaTeX `.\l` vs. `.\ll`)
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.).

10.3.2.2 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.

10.3.2.2.1 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-Spezifikationsprache 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.

10.3.2.2.2 Eigenschaften von SGML

SGML einfach als Markup-Sprache zu bezeichnen, ist eine völlig unzureichende Beschreibung der Möglichkeiten dieser Sprache:

- Aufgrund der Ausdruckstärke kann SGML auch als Datenbank-Definitionsprache 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.

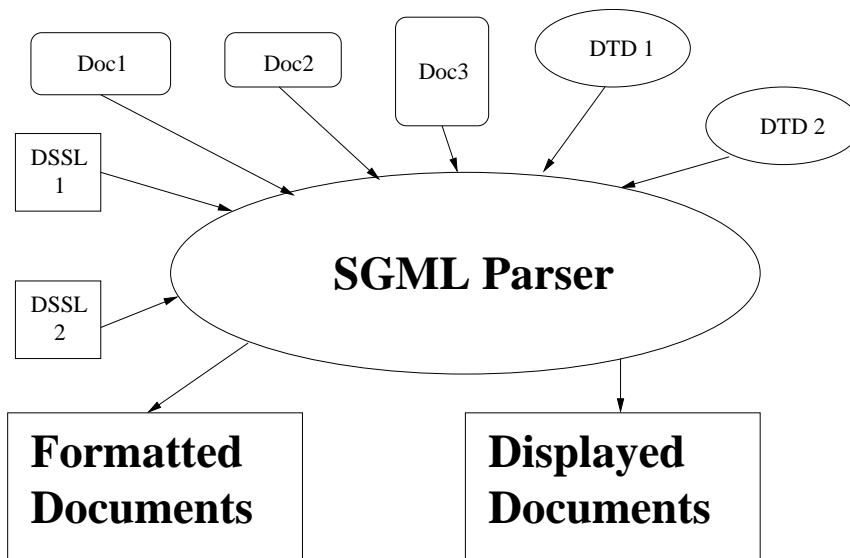


Abbildung 10.7: Verarbeitung von SGML-Dokumenten

10.3.2.2.3 Verarbeitung von SGML-Dokumenten

Abbildung 10.7 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 indiziert 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 darstellen zu können, benötigt man zusätzlich eine DSSSL-Spezifikation (sowie entsprechende Software); für unterschiedliche Ausgabemedien kann es verschiedene solcher Spezifikationen geben.

10.3.2.2.4 SGML-Markup

SGML unterstützt vier Arten von Markup:

1. deskriptiv durch Tags,
2. referentiell 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 Inhaltsmodell in Form einer Grammatik-Produktion, enthalten also selbst wieder andere Elemente. Das oberste Element ist jeweils das Dokument.

Abbildung 10.8 zeigt eine Beispiel-DTD und Abbildung 10.9 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 Elementes in Form eines regulären Ausdrucks notiert. #PCDATA steht dabei für beliebigen Text (als Inhalt eines Elements). Die Attributlisten der einzelnen Elemente werden

```

<!ELEMENT article - - (front, body) >
<!ELEMENT front 0 0 title author+>
<!ELEMENT title - - #PCDATA>
<!ELEMENT author - - #PCDATA>
<!ELEMENT body 0 0 section+>
<!ELEMENT section - 0 sectitle par*>
<!ELEMENT sectitle - - #PCDATA>
<!ELEMENT par - 0 #PCDATA>
<!ATTLIST article
      crdate CDATA #REQUIRED
      writer CDATA #REQUIRED>
<!ATTLIST body
      numbering CDATA #IMPLIED
      headings CDATA #IMPLIED>

```

Abbildung 10.8: Beispiel einer SGML-DTD

```

<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 10.9: Beispiel-Dokument zur Beispiel-DTD

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 10.10 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.

10.3.2.3 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-Funktionalität 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 Struktur und Layout aufgegeben. Zudem sind die logischen Tags auf ein Minimum beschränkt (i.w. Titel, Paragraph, Aufzählungen), was keine vernünftige logische Strukturierung ermöglicht. Dies

```

<!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 10.10: DTD zu HTML (Auszug)

erweist sich 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 das Fehlen entsprechender Tags zur Auszeichnung von Formeln bedingt ist).

10.3.2.4 XML

Da man durch die steigenden Anforderungen an Web-Angebote bald die Nachteile von HTML erkannt hatte, die auch durch immer neue HTML-Versionen nicht behoben werden konnten, kam es zur Rückbesinnung von SGML. Aufgrund der Erfahrungen mit SGML entschloss man sich jedoch, dies nicht direkt zu verwenden, sondern eine verbesserte Fassung hiervon zu entwickeln, die sogenannte Extended Markup Language (XML). XML ist im wesentlichen eine Vereinfachung von SGML und unterscheidet sich von diesem in folgenden Punkten:

- Start- und Ende-Tags müssen immer angegeben werden
- Eine Spezialform stellt das kombinierte Start-Ende-Tag dar, z.B. `
` (Zeilenumbruch in HTML) oder `` (Einbinden einer Abbildung).
- Da es für bestimmte offene Anwendungen nicht möglich ist, eine DTD anzugeben, erlaubt XML auch Dokumente ohne DTD. **Well-formed XML** bezeichnet solche syntaktisch korrekte XML-Dokumente (jedes Start-Tag ist durch ein Ende-Tag abgeschlossen). **Valid XML** setzt dagegen die Angabe der zugehörigen DTD voraus, die das betreffende XML-Dokument auch erfüllen muss.
- Bei Elementnamen wird zwischen Groß- und Kleinschreibung unterschieden, zudem sind, Unterstrich und Doppelpunkt in solchen Namen erlaubt. Letztere dienen vor allem zur Angabe von Präfixen zwecks Verwendung mehrerer Namespaces (Definitionen aus verschiedenen Quellen).
- Schließlich sind zahlreiche Sonderfälle aus SGML in XML verboten.

Abbildung 10.11 zeigt die XML-Variante zur DTD aus Abbildung 10.8, ein Dokument hierzu ist in Abbildung 10.12 dargestellt.

```
<!ELEMENT article (title, abstract, section+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT section
((title, body+) | (title, body*, subsectn+))>
<!ELEMENT subsectn (title, body+)>
<!ELEMENT body (figure | paragr)>
<!ELEMENT figure EMPTY>
<!ELEMENT paragr (#PCDATA)>

<!ATTLIST article author CDATA #REQUIRED
status (final | draft) "draft">
<!ATTLIST figure file ENTITY #IMPLIED>

<!ENTITY file SYSTEM "/tmp/picture.ps") NDATA postscript>
<!ENTITY amp "&">
```

Abbildung 10.11: Beispiel einer XML-DTD

Im Unterschied zu SGML hatte man bei der Definition von XML zwei unterschiedliche Arten von Anwendungen im Auge:

1. Strukturierte Dokumente (wie bei SGML): Mittlerweile wurden bereits DTDs für Dokumente aus speziellen Bereichen definiert, so z.B. MathML für mathematische Texte, CML für Dokumente aus der Chemie und SMIL für multimediale Dokumente.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE article SYSTEM "/services/dtds/article.dtd">
<article status = draft"
      author = "Cluet Christophides">
<title>From Structured Documents to ...</title>
<abstract>Structured Documents (e.g SGML) can
benefit from... </abstract>
<section>
<title>Introduction</title>
<body><paragr>This Paper is organized as follows.
...
</body></section>
<section>
<title>SGML preliminaries</title>
<body>
<figure/>
</body></section>
</article>

```

Abbildung 10.12: Beispiel-Dokument zur XML-DTD

2. Formatierte Daten: Hier dient XML als Syntax zur Serialisierung strukturierter Datenbestände (um diese in standardisierter Form zwischen unterschiedlichen Anwendungen im Web auszutauschen), wie z.B. für Spreadsheets, Metadaten oder Datenbanken.

Um beider Arten von Anwendungen zu unterstützen, gibt es eine ganze Reihe von zusätzlichen Standards (die sich zum großen Teil noch in der Entwicklung befinden):

XSL (XML Style Language) ist das Äquivalent zu DSSSL, dient also zur Definition von Stylesheets zur Präsentation. Bisher ist nur XSLT standardisiert, das lediglich die Transformation zwischen XML-Dokumenten erlaubt. Allerdings ist damit schon die Abbildung nach HTML möglich, und somit die Präsentation.

XLink standardisiert die Definition von Hypertext-Links. Links können sowohl intern als auch extern (außerhalb des XML-Dokumentes) abgelegt werden, eine Typisierung ist möglich, ein Link kann mehrere Ursprünge und mehrere Ziele haben, und es sind unterschiedliche Aktionen zum Verfolgen eines Links spezifizierbar.

XPointer spezifiziert Adressierungsmechanismen für XML, um Anker (Ursprünge/Ziele) in XLink anzugeben. Prinzipiell können beliebige Teile von XML-Dokumenten adressiert werden.

XML query language befindet sich derzeit noch in der Entwicklung. Vorläufig erlaubt **XPath** die Suche nach Teilbäumen von XML-Dokumenten. Die XML Query Language soll dagegen wesentlich ausdrucksstärker sein, um sowohl Textretrieval als auch Datenbanksuche zu unterstützen.

10.4 Zugriffspfade

10.4.1 Scanning

10.4.1.1 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.

10.4.1.2 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 Zeichenkette² 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:

$$E(t) = \frac{n - m + 1}{c^m}. \quad (10.1)$$

²Da diese Annahme in der Realität kaum zutrifft, werden außerdem experimentelle Resultate mit englischem Text präsentiert.

10.4.1.3 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 Java sieht wie folgt aus:

```

public void naivesearch(char[] text, int n, char[] pat, int m) {           /* Search pat[1..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:

$$\bar{C}_{firstmatch} = \frac{c^{m+1}}{c-1} - \frac{c}{c-1} \quad (10.2)$$

- Erwartungswert für die Gesamtzahl der Vergleiche:

$$\bar{C}_n = \frac{c}{c-1} \left(1 - \frac{1}{c^m} \right) (n - m + 1) + O(1) \quad (10.3)$$

(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.

10.4.1.4 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**:

aababrabr
abracadabra

```
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:

$$\text{next}[j] = \max\{i \mid (\text{pattern}[k] = \text{pattern}[j - i + k] \quad \text{für } k = 1, \dots, i - 1) \wedge \text{pattern}[i] \neq \text{pattern}[j]\}$$

oder

$$\begin{aligned} \text{next}[j] &= \max(\{0\} \cup M[j]) \quad \text{mit} \\ M[j] &= \{i \mid 1 \leq i < j \wedge \text{pattern}[i] \neq \text{pattern}[j] \wedge \\ &\quad \forall k \ 1 \leq k < i \ (\text{pattern}[k] = \text{pattern}[j - i + k])\} \end{aligned}$$

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 verglichen wird; **next[m+1]** definiert den Aufsetzpunkt im Fall eines Treffers.

Der in Java implementierte Algorithmus sieht wie folgt aus:

```
kmpsearch(char[] text, int n, char[] pat, int m) { /* Search pat[1..m] in text[1..n] */
  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;
  while (k <= n) { /* 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;
    }
  }
  pat[m+1] = END_OF_STRING;
}
```

Als obere Schranke für den Erwartungswert der Gesamtzahl der Vergleiche ergibt sich im Falle großer ($c > 10$) Alphabete:

$$\frac{\bar{C}_n}{n} \leq 1 + \frac{1}{c} - \frac{1}{c^m} \quad (10.4)$$

Im Vergleich zum naiven Algorithmus ergibt sich folgende Verringerung des Aufwands beim KMP-Algorithmus:

$$\frac{KMP}{naive} \approx 1 - \frac{2}{c^2} \tag{10.5}$$

10.4.1.5 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:

..xaxrxxxxxabracadabra

```

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 *iten* 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:

$$dd[j] = \min\{s + m - j \mid s \geq 1 \wedge ((s \geq j \vee pattern[j - s] \neq pattern[j]) \wedge ((s \geq i \vee pattern[i - s] = pattern[i]) \text{ für } j < i \leq m))\}$$

oder

$$dd[j] = \min\{s + m - j \mid 1 \leq s \leq m \wedge \forall i \ 0 \leq i \leq m - j \ (s + i < j \rightarrow (i = 0 \wedge pattern[j] \neq pattern[j - s] \vee i > 0 \wedge pattern[j] = pattern[j - s + i]))\}$$

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**:

$$d[x] = \min\{s \mid s = m \vee (0 \leq s < m \wedge pattern[m - s] = x)\}$$

Beispiel: Tabelle d für das Pattern abracadabra:

$$d['a'] = 0 \quad d['b'] = 2 \quad d['c'] = 6 \quad d['d'] = 4 \quad d['r'] = 1 \tag{10.6}$$

(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 Java sieht dann so aus:

```

public void bmsearch(char[] text, int n, char [] pat, int m) { /* Search pat[1..m] in text[1..n] */
    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

$$\frac{\bar{C}_n}{n} \geq \frac{1}{m} + \frac{m(m+1)}{2m^2c} + O(c^{-2}) \quad (10.7)$$

- Bei ungleicher Auftretenswahrscheinlichkeit der Zeichen gilt $\bar{C}_n/n < 1$ unter der Voraussetzung

$$c \left(1 - \sum_{i=1}^c p_i^2 \right) > 1. \quad (10.8)$$

10.4.1.6 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**:

..xaxrbdabracadabra

```

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:

$$d[x] = \min\{s \mid s = m \vee (1 \leq s < m \wedge pattern[m-s] = x)\}$$

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(char[] text, int n, char[] pat, int m) {
    int i, j, k;
    int d[MAX_ALPHABET_SIZE];

    for(j=0; j<MAX_ALPHABET_SIZE; j++)
        d[j] = m;
    for(j=1; j<m; j++)
        d[pat[j]] = m-j;

    pat[0] = CHARACTER_NOT_IN_THE_TEXT;

    text[0] = CHARACTER_NOT_IN_THE_PATTERN;
    i = m;
    while(i <= n) {
        k = i;
        for(j=m; text[k] == pat[j]; j--)
            k--;
        if (j == 0)
            Report_match_at_position(k+1);
        i += d[text[i]];
    }
}

```

Der asymptotischer Aufwand dieses Algorithmus' für n und c (mit $c \ll n$ und $m > 4$) ist

$$\frac{\tilde{C}_n}{n} = \frac{1}{m} + \frac{m+1}{2mc} + O(c^{-2}). \quad (10.9)$$

10.4.1.7 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 $1, \dots, i$ des Patterns und den Positionen $(j - i + 1), \dots, j$ 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:

$$state = \sum_{i=0}^{m-1} s_{i+1} \cdot 2^i \quad (10.10)$$

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:

$$state = (state \ll 1) \text{ or } T[currchar] \quad (10.11)$$

(Hierbei steht \ll 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:

$$T_x = \sum_{i=0}^{m-1} \delta(pat_{i+1} = x) \cdot 2^i \quad (10.12)$$

mit $\delta(C) = 0$, falls die Bedingung C erfüllt ist (sonst 1)

Beispiel für die Suche nach $ababc$ im Text $abdabababc$:

```
Text      :   a     b     d     a     b     a     b     a     b     c
T[x]      : 11010 10101 11111 11010 10101 11010 10101 11010 10101 01111
shift     : 11110 11100 11010 11110 11100 11010 10100 01010 10100 01010
shift-or: 11110 11101 11111 11110 11101 11010 10101 11010 10101 01111
```

Eine einfache Implementierung des Shift-Or-Algorithmus in Java ist nachfolgend angegeben.

```
public static void sosearch(char[] text, int n, char[] pat, int m) {
    /* Search pat[1..m] in text[1..n] */
    int start, end, current;
    long state, lim, mbits, hit;
    long MAXSYM = 65535;
    long WORD = 62;
    long T[] = new long[MAXSYM];
    int i, j;

    if(m > WORD) System.out.println("abort - use pat size <= word size");

    state = 1; for(j=m; j>0; j--) state *= 2; state -= 1;
    for(i=0; i<MAXSYM; i++) T[i] = state;

    /* for pat[j] in T[pat[j]] set bit j to 0 */
    mbits = state;
    lim = (state << 1) & mbits;

    /* state vector for the characters of the pattern */
    for(j=0; j<m; j++, lim = ((lim << 1) & mbits) + 1) T[pat[j]] &= lim;

    hit = (state + 1)/2;

    /* pattern search */
    for(i=0; i<=n-m+1; i++){
        state = ((state << 1) | T[text[i]]) & mbits;
        if(state < hit) System.out.println("match at position " + (i-m+1));
    }
}
```

Die Komplexität dieses 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.

10.4.1.8 Erweiterungen des Shift-Or-Algorithmus'

Im folgenden beschreiben wir die in [Baeza-Yates & Gonnet 92] vorgestellten Erweiterungen des Shift-Or-Algorithmus'.

10.4.1.8.1 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:

$$T_x = \sum_{i=0}^{m-1} \delta(pat_{i+1} \in Class_{i+1}) \cdot 2^i$$

Somit ändert sich nur die Präprozessierung des Patterns, der eigentliche Algorithmus bleibt unverändert.

10.4.1.8.2 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 abdabababc 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
shift	:	99990	99900	99010	91210	22200	23010	40200	03010	40200	03010
shift-or:	:	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.

10.4.1.8.3 Alternative Patterns

Hierbei wird gleichzeitig nach mehreren Patterns p_1, \dots, 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 $O(\lceil \frac{m_{max}}{w} \rceil l n)$.
- 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.

10.4.1.9 Vergleich der Verfahren

Abbildung 10.13 zeigt experimentelle Ergebnisse für die Anwendung der vorgestellten Algorithmen auf englischsprachige Texte.

10.4.2 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].

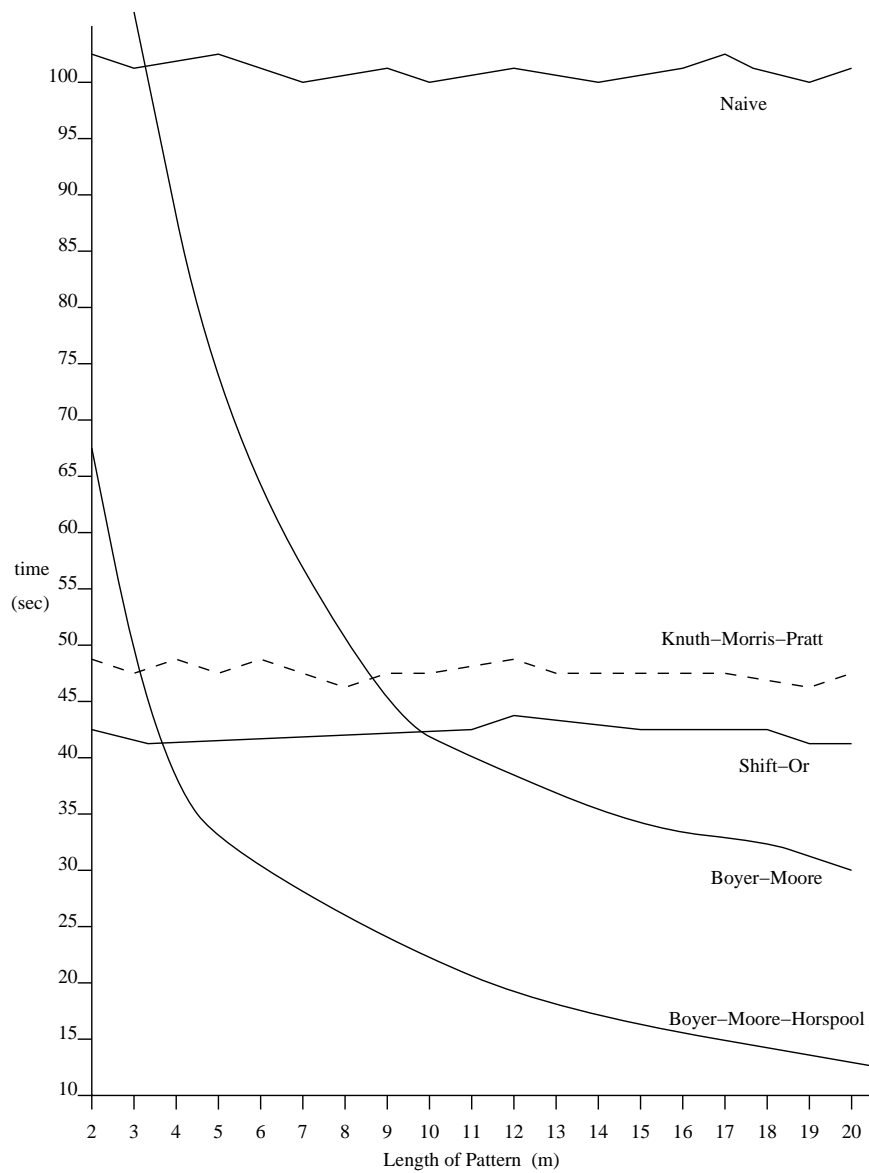


Abbildung 10.13: Experimentelle Ergebnisse für englischsprachigen Text

10.4.2.1 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“.

10.4.2.2 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:

$$f(0, 0) = 0 \quad (10.13)$$

$$f(i, j) = \min\{f(i-1, j) + 1, f(i, j-1) + 1, f(i-1, j-1) + d(s_i, t_j), \\ f(i-2, j-2) + d(s_{i-1}, t_j) + d(s_i, t_{j-1}) + 1\} \quad (10.14)$$

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.

10.4.2.3 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.

10.4.3 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.

10.4.3.1 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}	\dots
t_2 :	d_5	d_{15}	d_{89}	\dots	

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}	\dots
-------	-------	----------	----------	----------	---------

Beispiel: Ergebnisliste für $t_1 \wedge t_2$

d_{15}	d_{89}	\dots
----------	----------	---------

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 abzurufen, 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.

10.4.3.2 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 einen einfachen Algorithmus 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.

Im folgenden wird ein einfacher zwei Algorithmus vorgestellt, der die oben gestellten Anforderungen erfüllt. Effizientere Algorithmen sind in [Moffat & Zobel 96] beschrieben (siehe die zugehörigen Folien zum nächsten Abschnitt).

10.4.3.2.1 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.

Als Beispiel betrachten wir wieder die invertierten Listen zu zwei Termen t_1 und t_2 :

t_1	d_2, u_{12}	d_{15}, u_{115}	d_{23}, u_{123}	d_{89}, u_{189}	\dots
t_2	d_5, u_{25}	d_{15}, u_{215}	d_{89}, u_{289}	\dots	

Mit zugehörigen Fragetermgewichten w_1 und w_2 und dem Skalarprodukt als Retrievalfunktion erhalten wir dann folgendes Ergebnis:

$$d_2: w_1 \cdot u_{12}, d_5: w_2 \cdot u_{15}, d_{15}: w_1 \cdot u_{15} + w_2 \cdot u_{15}, d_{23}: w_1 \cdot u_{123}, d_{89}: w_1 \cdot u_{189} + w_2 \cdot u_{189}$$

Der nachstehende Algorithmus verarbeitet die einzelnen Listen nacheinander — dadurch wird der Hauptspeicherbedarf minimiert.

1. Für jedes Dokument der Kollektion: Setze Akkumulator A_d auf 0
2. Für jeden Term der Anfrage:
 - (a) Hole I_t , die invertierte Liste für t .
 - (b) Für jedes Paar \langle Dokumentnummer d , Indexierungsgewicht $u_{d,t}$ \rangle in I_t setze $A_d \leftarrow A_d + w_{q,t} \cdot u_{d,t}$.
3. Bestimme die k höchsten Werte A_d
4. Für jedes dieser k Dokumente d :
 - a) Hole die Adresse von Dokument d .
 - b) Hole Dokument d and präsentiere es dem Benutzer.

10.4.3.3 Komprimierung invertierter Listen

Siehe [Moffat & Zobel 96].

10.4.4 Signaturen

10.4.4.1 Das Signaturkonzept

10.4.4.1.1 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

$$S := \langle b_1, b_2, \dots, b_L \rangle \quad \text{mit} \quad b_i \in \{0, 1\}, L \in \mathbb{N} \quad (10.15)$$

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 eindeutige Abbildung zwischen Wörtern und Signaturen existieren, können Signaturen nur als Filter eingesetzt werden, um eine Obermenge der gesuchten Wörter bzw. Dokumente zu bestimmen (siehe 10.4.4.1.2).

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

$$S \supseteq_S S^Q \Leftrightarrow (\forall)((1 \leq i \leq L) \wedge ((b_i^Q = 1) \Rightarrow (b_i = 1))), S, S^Q \in S_L. \quad (10.16)$$

Dieser Operator kann auf effiziente Bitoperationen zurückgeführt werden, wie sie von allen Rechnerarchitekturen angeboten werden:

$$S \supseteq_S S^Q \Leftrightarrow S \wedge S^Q = S^Q \Leftrightarrow (\neg S) \wedge S^Q = 0_S \quad (10.17)$$

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
-------------	--------------

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.

10.4.4.1.2 Prinzipielle Organisation eines Signatur-Systems

Abb. 10.14 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.

10.4.4.2 Codierungsmethoden

Wir betrachten nun verschiedene Methoden, um Mengen von Wörtern bzw. n-grams auf Signaturen abzubilden

10.4.4.2.1 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)

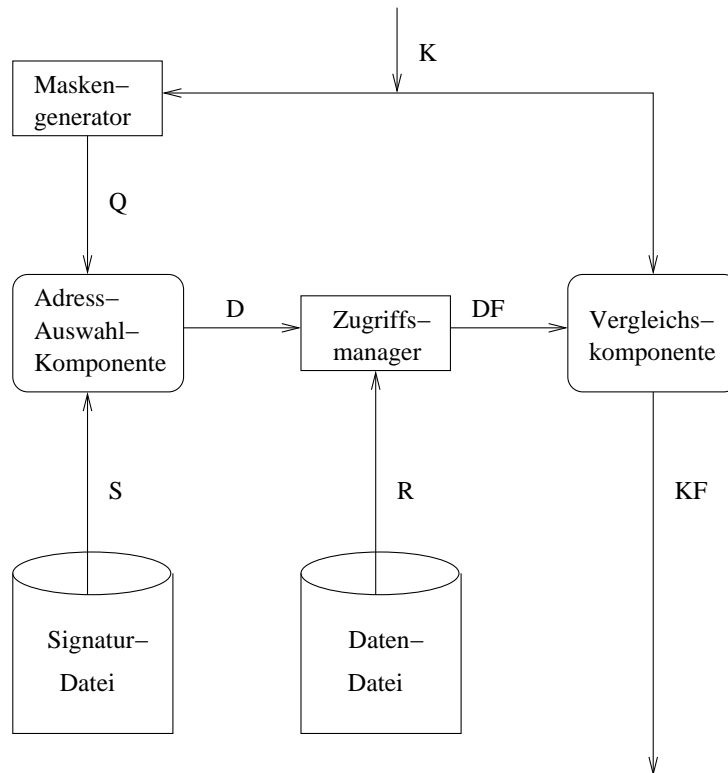


Abbildung 10.14: Organisation eines Signatur-Systems

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:

$$SP = \binom{L}{g} = \frac{L!}{g!(L-g)!} \tag{10.18}$$

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

$$SP_1 = SP(L, \lfloor \frac{L}{2} \rfloor) \quad \text{und}$$

$$SP_2 = SP(L, \lfloor \frac{L}{2} \rfloor - 1) = SP_1 \cdot \frac{g}{L - (g - 1)}.$$

Wegen $g \leq \frac{L}{2}$ folgt daraus:

$$SP_2 \leq SP_1 \cdot \left(\frac{\lfloor \frac{L}{2} \rfloor}{\lfloor \frac{L}{2} \rfloor + 1} \right) < SP_1$$

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 $SP = \lfloor \frac{L}{2} \rfloor$ 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

$$F = \left(\frac{W}{SP} - 1 \right) \frac{N}{W}. \tag{10.19}$$

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 10.19 erhält man für das Signaturpotential als Funktion der Fehlerrate, der Wörterbuchgröße und des Datenvolumens:

$$SP = \frac{W \cdot N}{F \cdot W + N}. \tag{10.20}$$

Über $SP = \binom{L}{g}$ kann man daraus die Signaturlänge bestimmen. Tabelle 10.1 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 10.1: 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 verketteten (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

$$E(V(x)) = B \left(1 - \left(1 - \frac{1}{B} \right)^x \right). \tag{10.21}$$

($1 - 1/B$ 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 10.21 $x = B \frac{w}{SP}$ 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**:

$$\begin{aligned} f &\approx 1 - \exp \left(-\frac{w}{SP} \right) \\ &\approx \frac{w}{SP} \end{aligned}$$

10.4.4.2.2 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]:

$$p(L, g, \lambda, t) = \sum_{j=1}^t (-1)^j \binom{t}{j} \left(\frac{\binom{L-j}{g}}{\binom{L}{g}} \right)^\lambda \quad (10.22)$$

Für kleine t und λ läßt sich diese Formel annähern durch:

$$p(L, g, \lambda, t) \approx [p(L, g, \lambda, 1)]^t = \left(1 - \left(1 - \frac{g}{L}\right)^\lambda\right)^t \quad (10.23)$$

Die Herleitung dieser Approximation kann man dadurch 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

$$\gamma = \sum_{i=1}^L p(1) = L \cdot \left(1 - \left(1 - \frac{g}{L}\right)^{\bar{a}}\right) \quad (10.24)$$

in die Approximation der Fehlerwahrscheinlichkeit

$$f \approx \left(1 - \left(1 - \frac{g}{L}\right)^\lambda\right)^\gamma \quad (10.25)$$

ein. Nach Umformung zu

$$\ln f \approx L \cdot \left(1 - \left(1 - \frac{g}{L}\right)^\lambda\right)^{\bar{a}} \cdot \ln \left(1 - \left(1 - \frac{g}{L}\right)^\lambda\right)$$

und Ersetzung von

$$\left(1 - \frac{g}{L}\right)^{\bar{a}} \quad \text{durch} \quad \exp\left(\frac{\bar{a}}{\lambda} \cdot \ln \left(1 - \frac{g}{L}\right)^\lambda\right) \quad (10.26)$$

kann man $y = \left(1 - \frac{g}{L}\right)^\lambda$ setzen und erhält

$$\ln f \approx L \cdot \left(1 - \exp\left(\frac{\bar{a}}{\lambda} \ln y\right)\right) \cdot \ln(1 - y) \quad (10.27)$$

Um die nun folgende Minimierung dieser Funktion zu vereinfachen, setzen wir

$$e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} \quad \text{mit} \quad x = \frac{\bar{a}}{\lambda} \quad (10.28)$$

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

$$\ln\left(\frac{F}{N}\right) \approx -L \cdot \frac{\bar{a}}{\lambda} \cdot \left(\ln \frac{1}{2}\right)^2$$

und somit die optimale Signaturlänge

$$L_{opt} = \frac{\lambda}{\bar{a} \cdot \ln 2} \cdot \log_2 \frac{N}{F} \quad (10.29)$$

Aus $y = \frac{1}{2} = \left(1 - \frac{g}{L}\right)^\lambda$ resultiert schließlich das **optimale Wortgewicht**

$$g_{opt} = L \cdot \left(1 - 2^{-\frac{1}{\lambda}}\right). \quad (10.30)$$

Daß man mit g_{opt} richtig liegt, sieht man, indem man den Erwartungswert des Satzsignaturgewichts Γ aus den Wortsignaturen ermittelt:

$$\Gamma = \sum_{i=1}^L p(g_{opt}) = L \cdot \left(1 - \left(1 - \frac{g_{opt}}{L}\right)^\lambda\right) = \frac{L}{2}$$

Dem $\frac{L}{2}$ ist das optimale Satzsignaturgewicht nach Überlagerung von λ Wortsignaturen mit optimalem Gewicht g_{opt} .

³Das entspricht bei Verwendung der durch die Gleichungen 10.29 und 10.30 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.

Wir betrachten nun die Abhängigkeit der Fehlerwahrscheinlichkeit von der Anzahl Wörter in der Anfrage. Setzt man g_{opt} in die Ausgangsformel 10.25 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:

$$t = g(a_q) = \frac{\lambda}{\bar{a} \cdot \ln 2} \cdot \log_2 \frac{N}{F} \cdot \left(1 - 2^{-\frac{a_q}{\bar{a}}}\right).$$

Für $a_q = \bar{a}$ erhalten wir die gewünschte Fehlerrate $f(g(a_q)) = F/N$. Wesentlich wichtiger ist aber der allgemeine Zusammenhang 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.

10.4.4.2.3 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-grams⁴ für einen Substring der n-gram-Anzahl ν_Q die Fehlerwahrscheinlichkeit

$$f(\nu_Q) = \left(1 - \left(1 - \frac{1}{L}\right)^\nu\right)^\gamma \quad \text{mit} \quad \gamma = L \cdot \left(1 - \left(1 - \frac{1}{L}\right)^{\nu_Q}\right). \quad (10.31)$$

⁴Unter 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.

Das erwartete Gewicht einer Stringsignatur liegt bei

$$\Gamma = \sum_{i=1}^L f(i) = L \cdot \left(1 - \left(1 - \frac{1}{L} \right)^\nu \right).$$

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:

$$\begin{aligned} f(\nu_Q) &\approx \left(1 - \left(1 - \frac{1}{L} \right)^\nu \right)^\gamma \\ &\approx \left(1 - \left(1 - \frac{\nu}{L} \right) \right)^{\nu_Q} \\ &= \left(\frac{\nu}{L} \right)^{\nu_Q} \end{aligned} \quad (10.32)$$

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

$$L = \nu \cdot \left(\frac{N}{F} \right)^{\frac{1}{\nu_Q}} \quad (10.33)$$

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 infolgedessen für Suchstrings, die kürzer als n Zeichen sind, alle Datensätze sequentiell durchsucht werden müssen.

Die oben erwähnte empirische Streuung von n-grams in Texten wurde für die deutsche Sprache bereits von [Bauer & Goos 82] mit dem in Tabelle 10.2 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

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 10.2: Empirische Streuung von n-grams in deutschsprachigen Texten

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.

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 10.3: Abbildungstabelle für Buchstaben

Praktisch wird das anhand der Buchstabentabelle 10.3 deutlich, die für die Strings der beiden Klassen {‘er’, ‘en’} und {‘ee’} unter der Annahme einer unabhängigen 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

$$\begin{aligned}
 T(n - \text{gram}) &= T(t - \text{gram}_{\frac{n}{t}-1}, \dots, t - \text{gram}_0) \\
 &= \sum_{i=0}^{\frac{n}{t}-1} c^i \cdot I(t - \text{gram}_i).
 \end{aligned}
 \tag{10.34}$$

Dabei bildet $I(t - \text{gram})$ jedes t-gram auf einen der c Klassenindices $0 \dots c - 1$ 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.

10.4.4.3 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.

10.4.4.3.1 Sequentielle Signaturen

	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 10.15: Beispiel für sequentielle Signaturen

Abbildung 10.15 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.

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 $\lceil \frac{L}{8} \rceil + size_{@}$. Daraus ergibt sich die Anzahl Einträge pro Seite:

$$K = \left\lfloor \frac{size_p}{\lceil \frac{L}{8} \rceil + size_{@}} \right\rfloor \tag{10.35}$$

Wir betrachten nun für verschiedene Datenbank-Operationen die **Anzahl Seitenzugriffe**. Für die **Retrieve-Operation** müssen alle Signaturseiten sequentiell gelesen werden:

$$Seq^R = \left\lceil \frac{N}{K} \right\rceil + F + D \tag{10.36}$$

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:

$$Seq^I = 2 + 2 = 4 \tag{10.37}$$

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.

$$Seq^D = \left\lceil \frac{N}{2 \cdot K} \right\rceil + 1 + 2 = \left\lceil \frac{N}{2 \cdot K} \right\rceil + 3 \tag{10.38}$$

Der gesamte **Speicherplatzbedarf** (in Seiten) für die Datenbank mit sequentiellen Signaturen ist

$$Seq^S = \left\lceil \frac{N}{K} \right\rceil + M. \tag{10.39}$$

10.4.4.3.2 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 10.16: Beispiel zur Bitscheibenorganisation

Eine wesentlich effizientere Organisationsform für Signaturen ist die Bitscheibenorganisation, wie sie beispielhaft in Abbildung 10.16 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 10.16 folgende Ergebnisbitliste:

$$\begin{aligned}
 b_r^* &= b_1^* \wedge b_3^* \\
 &= \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \wedge \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}
 \end{aligned}$$

Allgemein berechnet man die Ergebnisbitliste nach der Formel:

$$b_r^* = \bigwedge_{j=q_1}^{q_\gamma(S_Q)} b_j, \quad q_i \in \{q | (1 \leq q \leq L) \wedge b_q^Q = 1\} \tag{10.40}$$

Die Menge der Adressen der Trefferkandidaten ist dann $R = \{i | b_r^i = 1\}$.

Bei disjunktive Anfragen der Form $S(Q_1) \vee S(Q_2) \vee \dots \vee S(Q_T)$ 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

$$BS^R = T \cdot \left(\gamma(Q) \left\lceil \frac{N}{8 \cdot size_p} \right\rceil + Z + F + D \right) \tag{10.41}$$

Hierbei bezeichnet Z die Anzahl Seitenzugriffe auf die Zuordnungstabelle. Die Gesamtzahl Seiten der Zuordnungstabelle ist

$$R = \left\lceil \frac{N}{\frac{size_p}{size_{\text{®}}}} \right\rceil. \tag{10.42}$$

Damit läßt sich dann der Erwartungswert von Z berechnen zu:

$$Z = R \cdot \left(1 - \left(1 - \frac{1}{R} \right)^{(F+D)} \right) \tag{10.43}$$

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 Grenzwert 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 Signaturngewicht des einzufügenden Datensatzes, dann müssen $\gamma(S)$ Bitscheibenseiten, der Eintrag in der Zuordnungstabelle und die Datenseite geändert werden:

$$BS^I = 2 \cdot \gamma(S) + 2 + 2. \quad (10.44)$$

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:

$$BS^D = \gamma \cdot \left(\left\lceil \frac{N}{8 \cdot \text{size}_p} \right\rceil + 1 \right) + Z + 2. \quad (10.45)$$

Der **Speicherplatzbedarf** (in Seiten) bei Bitscheibenorganisation beträgt

$$BS^S = L \cdot \left\lceil \frac{N}{8 \cdot \text{size}_p} \right\rceil + \left\lceil N \cdot \frac{\text{size}_@}{\text{size}_p} \right\rceil + M. \quad (10.46)$$

10.4.5 PAT-Bäume

Der PAT-Baum ist eine Datenstruktur zur effizienten Textsuche in großen Datenbeständen. Insbesondere kann auch die Suche in strukturierten Dokumenten effektiv unterstützt werden (was hier allerdings nicht betrachtet wird). Die nachfolgende Beschreibung orientiert sich im wesentlichen an der Darstellung in [Gonnet et al. 92].

10.4.5.0.3 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 darstellen, 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 $\blacksquare A SA \dots \blacksquare < \blacksquare AMP \dots \blacksquare < \blacksquare E ST \dots \blacksquare$).

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 absolut über 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

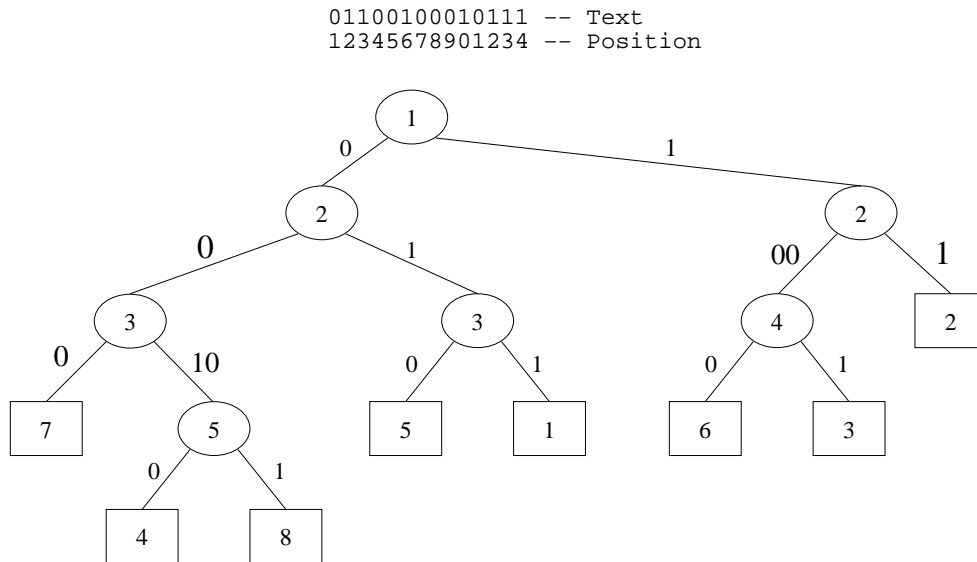


Abbildung 10.17: PAT-Baum nach dem Einfügen der ersten acht sistrings

den externen Knoten des Patricia-Baums gespeichert. Die internen Knoten enthalten nur die Angabe über das Verzweigungsbit sowie Zeiger auf die beiden Nachfolger. Ein Beispiel für einen PAT-Baum zeigt Abbildung 10.17.

10.4.5.0.4 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

10.4.5.0.4.1 Präfix-Suche

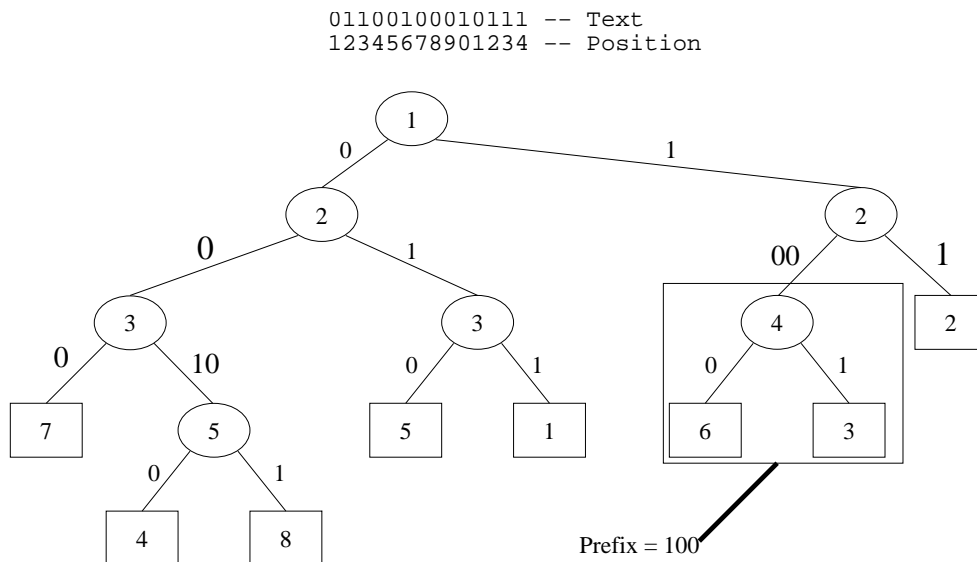


Abbildung 10.18: Präfix-Suche

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 10.18. Hier liefert die Suche nach dem Präfix 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.

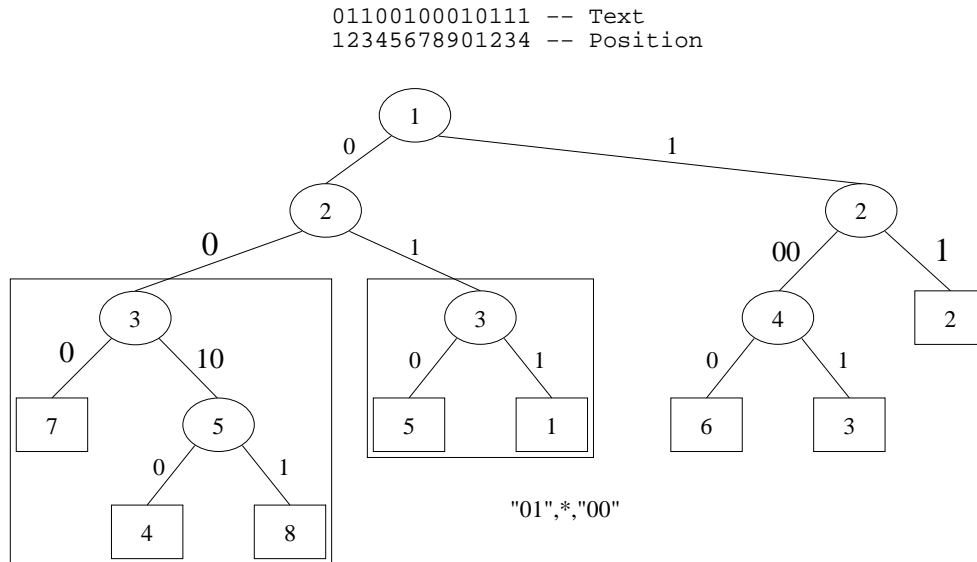


Abbildung 10.19: Reihenfolge-Suche

10.4.5.0.4.2 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 $O((m_1 + m_2) \log m_1)$

10.4.5.0.4.3 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 10.20. Der Zeitaufwand für diesen Algorithmus beträgt offensichtlich $O(\log n)$.

10.4.5.0.4.4 Längste Wiederholung

Sucht man nach dem längsten sistring, der zweimal im Text vorkommt, so muß man nur den höchsten internen Knoten im PAT-Baum bestimmen (wobei zusätzlich evtl. übersprungene Bits zu berücksichtigen sind). Diese Aufgabe kann bereits beim Aufbau des PAT-Baums gelöst werden. Ein Beispiel hierzu zeigt Abbildung 10.21.

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 müssen, 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 $O(\log n)$.

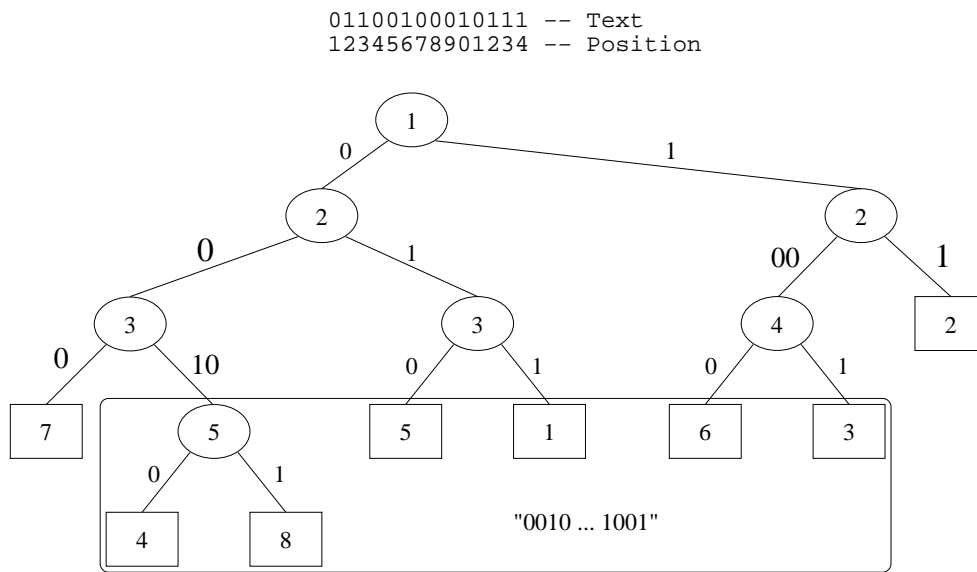


Abbildung 10.20: Bereichssuche

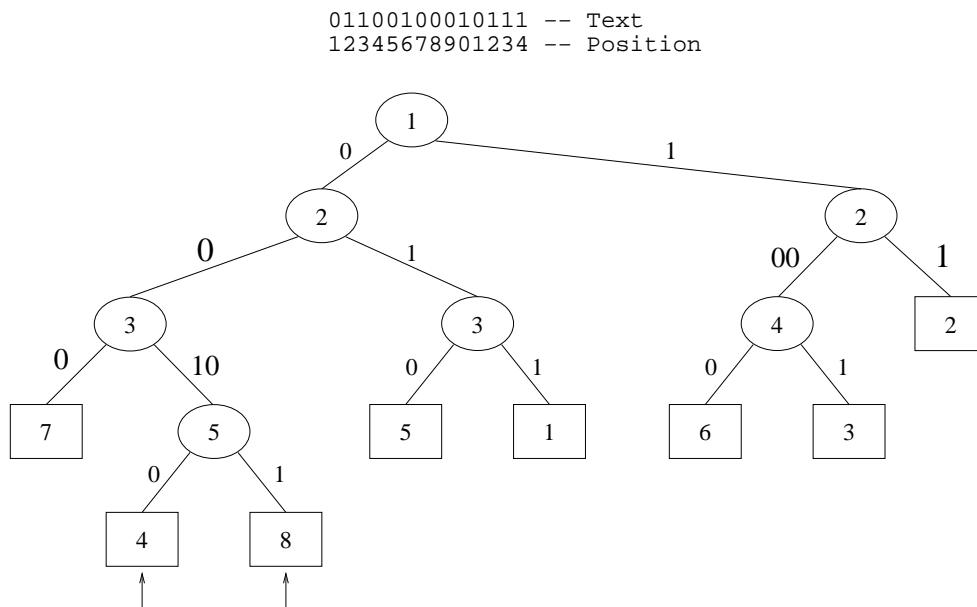


Abbildung 10.21: Suche nach der längsten Wiederholung

01100100010111 -- Text
 12345678901234 -- Position

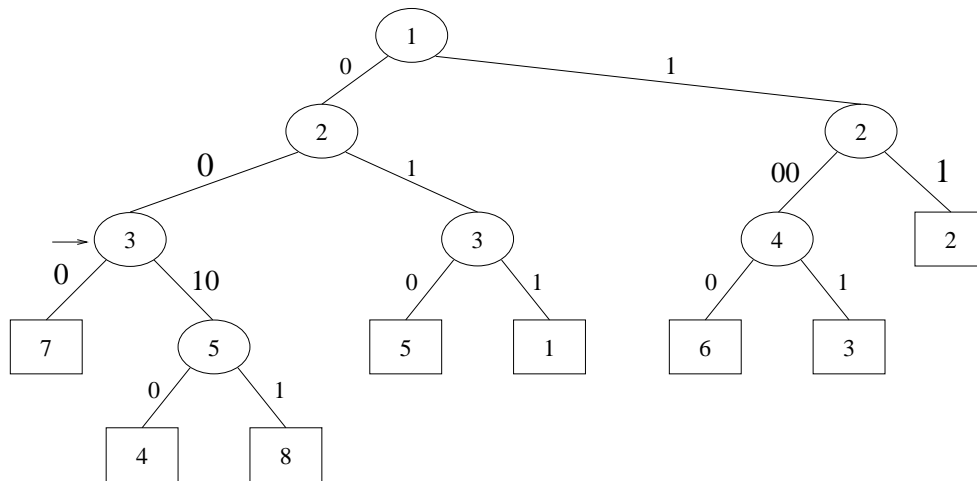


Abbildung 10.22: Suche nach dem häufigsten Bigram

10.4.5.0.4.5 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 10.22 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 der mit Blank beginnenden sistrings, und zudem wird beim Durchlaufen immer nur bis zum zweiten Blank gesucht.

10.4.5.0.4.6 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 10.23 zeigt ein Beispiel hierzu. (Zu den Details siehe [Gonnet et al. 92]).

10.4.5.0.5 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 entwickelt, 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 10.24. 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 $2 \log_2 n - 1$ Vergleiche und $4 \log_2 n$ Plattenzugriffe. Andere Suchoperationen werden hingegen aufwendiger.

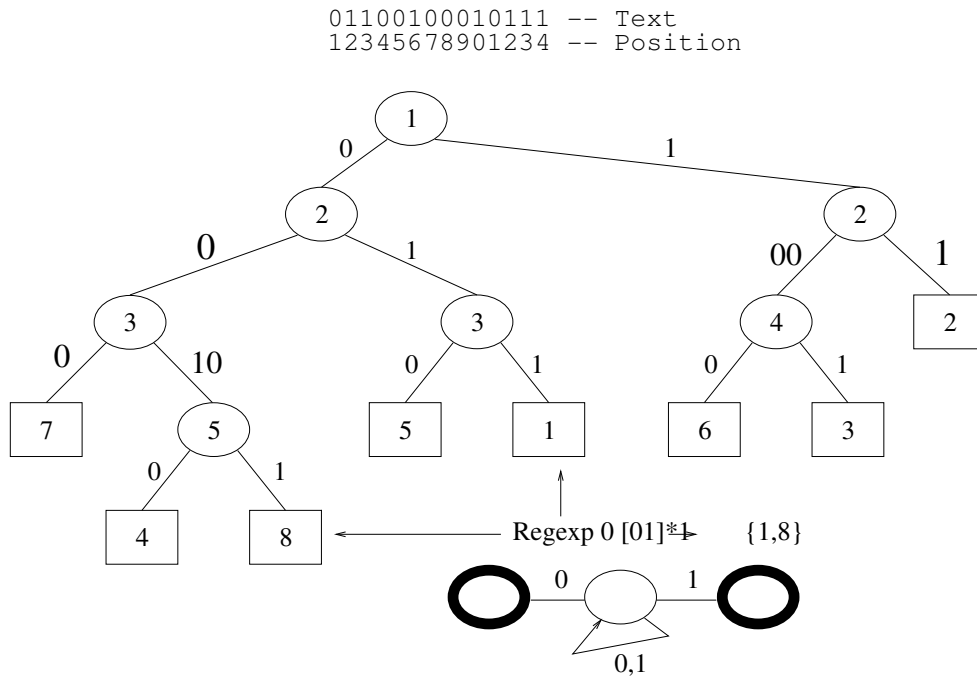


Abbildung 10.23: Suche nach regulären Ausdrücken

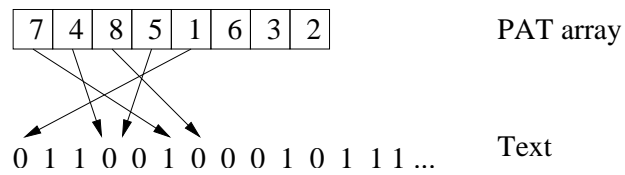


Abbildung 10.24: Beispiel eines PAT-Arrays

Literaturverzeichnis

- Agosti, M.; Crestani, F.; Pasi, G. (Hrsg.)** (2001). *Lectures in Information Retrieval*. Springer, Heidelberg et al.
- 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.; Ribeiro-Neto, B.** (1999). *Modern Information Retrieval*. Addison Wesley.
- Baeza-Yates, R.** (1989). Algorithms for String Searching: A Survey. *SIGIR Forum* 23(3/4), S. 34–58.
- Bates, M. J.** (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, Heidelberg et al.
- Belew, R.** (2000). *Finding Out About. A Cognitive Perspective on Search Engine Technology and the WWW*. Cambridge University Press, Cambridge, UK.
- Biebricher, P.; Fuhr, N.; Knorz, G.; Lustig, G.; Schwantner, M.** (1988). The Automatic Indexing System AIR/PHYS - from Research to Application. In: *11th International Conference on Research and Development in Information Retrieval*, S. 333–342. Presses Universitaires de Grenoble, Grenoble, France.
- 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.
- Brajnik, G.; Guida, G.; Tasso, C.** (1988). IR-NLI II: Applying Man-Machine Interaction and Artificial Intelligence Concepts to Information Retrieval. In: *11th International Conference on Research & Development in Information Retrieval*, S. 387–399. Presses Universitaires de Grenoble, Grenoble, France.
- Burkart, M.** (1990). Dokumentations-sprachen. In: *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, Heidelberg et al.
- Charniak, E.; Hendrickson, C.; Jacobson, N.; Perkowski, N.** (1993). Equations for Part-of-speech Tagging. In: *Proceedings of the Eleventh National Conference on Artificial Intelligence*, S. 784–789. Morgan Kaufman, Menlo Park, CA.
- Chiararella, Y.; Mulhem, P.; Fourel, F.** (1996). *A Model for Multimedia Information Retrieval*. Technischer Bericht, FERMI ESPRIT BRA 8134, University of Glasgow.
- Cleverdon, C.** (1991). The Significance of the Cranfield Tests on Index Languages. In: *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 3–11. ACM, New York.
- Codd, E. F.** (1986). Missing Information (Applicable and Inapplicable) in Relational Databases. *SIGMOD Record* 15(4), S. 53–78.

- Cooper, W. S.** (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: *Proceedings of the Fourteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 57–61. ACM, New York.
- Cooper, W.** (1995). Some Inconsistencies and Misidentified Modeling Assumptions in Probabilistic Information Retrieval. *ACM Transactions on Information Systems* 13(1), S. 100–111.
- Cox, D.** (1970). *Analysis of Binary Data*. Methuen, London.
- Crestani, F.; Lalmas, M.; van Rijsbergen, C. J.; Campbell, I.** (1998). “Is this document relevant? ... probably”: a survey of probabilistic models in information retrieval. *ACM Computing Surveys* 30(4), S. 528–552.
- Cutting, D.; Pedersen, J.; Karger, D.; Tukey, J.** (1992). Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In: *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 318–329. ACM, New York. <http://citeseer.nj.nec.com/cutting92scattergather.html>.
- Deppisch, U.** (1989). *Signaturen in Datenbanksystemen*. PhD thesis, TH Darmstadt, Fachbereich Informatik.
- Eastman, C.** (1989). Approximate Retrieval: A Comparison of Information Retrieval and Database Management Systems. *IEEE Data Engineering Bulletin* 12(2), S. 41–45.
- Faloutsos, C.** (1985). Access Methods for Text. *ACM Computing Surveys* 17(1), S. 49–74.
- Ferber, R.** (2003). *Information Retrieval. Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web*. dpunkt Verlag, Heidelberg.
- Fienberg, S.** (1980). *The Analysis of Cross-Classified Categorical Data*. MIT Press, Cambridge, Mass., 2. Auflage.
- Frakes, W.; Baeza-Yates, R.** (1992). *Information Retrieval. Data Structures & Algorithms*. Prentice Hall, Englewood Cliffs.
- Freeman, D.** (1987). *Applied Categorical Data Analysis*. Dekker, New York.
- Frei, H.; Meienberg, S.; Schäuble, P.** (1991). The Perils of Interpreting Recall and Precision Values. In: *Information Retrieval*, S. 1–10. Springer, Heidelberg 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.; Hütther, 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: *Research and Development in Information Retrieval*, S. 391–408. Cambridge University Press, Cambridge.
- Fuhr, N.** (1986). Rankingexperimente mit gewichteter Indexierung. In: *Deutscher Dokumentartag 1985*, S. 222–238. K.G. Saur, München, New York, London, Paris.
- Fuhr, N.** (1988). *Probabilistisches Indexing und Retrieval*. PhD thesis, 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.** (1992). Probabilistic Models in Information Retrieval. *The Computer Journal* 35(3), S. 243–255.
- Fuhr, N.** (1995). Modelling Hypermedia Retrieval in Datalog. In: Kuhlen, R.; Rittberger, M. (Hrsg.): *Hypertext - Information Retrieval - Multimedia, Synergieeffekte elektronischer Informationssysteme, Proceedings HIM '95*, Band 20 von *Schriften zur Informationswissenschaft*, S. 163–174. Universitätsverlag Konstanz, Konstanz.
- Gadd, T.** (1988). 'Fishing 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.

- Greene, B.; Rubin, G.** (1971). *Automatic Grammatical Tagging of English*. Technical report, Brown University, Providence, RI.
- Halpern, J. Y.** (1990). An Analysis of First-Order Logics of Probability. *Artificial Intelligence* 46, S. 311–350.
- Harman, D.** (1995). Overview of the Second Text Retrieval Conference (TREC-2). *Information Processing and Management* 31(03), S. 271–290.
- 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.
- Hiemstra, D.** (1998). A Linguistically Motivated Probabilistic Model of Information Retrieval. In: *Lecture Notes In Computer Science - Research and Advanced Technology for Digital Libraries - Proceedings of the second European Conference on Research and Advanced Technology for Digital Libraries: ECDL'98*, S. 569–584. Springer Verlag.
- Joachims, T.** (2001). *The Maximum-Margin Approach to Learning Text Classifiers. Methods, Theory, and Algorithms*. PhD thesis, Fachbereich Informatik, Universität Dortmund.
- 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: *Experimentelles und praktisches Information Retrieval*, S. 35–58. Universitätsverlag Konstanz, Konstanz.
- Krönert, G.** (1988). Genormte Austauschformate für Dokumente. *Informatik Spektrum* 11(11), S. 71–84.
- Kuhlen, R.** (1977). *Experimentelle Morphologie in der Informationswissenschaft*. Verlag Dokumentation, München.
- Kuhlen, R.** (1990). Zum Stand pragmatischer Forschung in der Informationswissenschaft. In: *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: *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.
- Lalmas, M.** (1997). Dempster-Shafer's Theory of Evidence Applied to Structured Documents: Modelling Uncertainty. In: Belkin, N. J.; Narasimhalu, A. D.; Willet, P. (Hrsg.): *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 110–118. ACM, New York.
- Lee, J. H.; Kim, W. Y.; Kim, M. H.; Lee, Y. J.** (1993). On the Evaluation of Boolean Operators in the Extended Boolean Retrieval Framework. In [SIG93], S. 291–297.
- 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.
- 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 [SIG93], S. 298–308.
- Moffat, A.; Zobel, J.** (1996). Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems* 14(4), S. 349–379.

- 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.
- Nie, J.** (1989). An Information Retrieval Model Based on Modal Logic. *Information Processing and 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, California.
- Peters, C. (Hrsg.)** (2001). *Cross-Language Information Retrieval and Evaluation*, Band 2069 von *Lecture Notes in Computer Science*, Heidelberg et al. Springer.
- Ponte, J.; Croft, W.** (1998). A Language Modeling Approach to Information Retrieval. In: Croft, W. B.; Moffat, A.; van Rijsbergen, C. J.; Wilkinson, R.; Zobel, J. (Hrsg.): *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 275–281. ACM, New York.
- Rabitti, R.; Zezula, P.** (1990). A Dynamic Signature Technique for Multimedia Databases. In: *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. 279–287.
- Raghavan, V. V.; Bollmann, P.; Jung, G. S.** (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*, Kapitel 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: *Proceedings of the Twelfth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 77–86. ACM, New York.
- Roberts, C.** (1979). Patial-Match Retrieval via the Method of Superimposed Coding. *Proceedings of the IEEE* 67(12), S. 1624–1642.
- 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.; Fuhr, N.** (1996). Retrieval of Complex Objects Using a Four-Valued Logic. In: *Proceedings of the 19th International ACM SIGIR Conference on Research and Development in Information Retrieval*, S. 206–214. ACM, New York.
- Rölleke, T.** (1994). *Equivalences of the Probabilistic Relational Algebra*. Technical report, University of Dortmund, Department of Computer Science.
- 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. (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.; Fox, E.; Wu, H.** (1983). Extended Boolean Information Retrieval. *Communications of the ACM* 26, S. 1022–1036.
- 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.
- (1993). *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York. ACM.
- Sparck Jones, K.; Willet, P.** (1997). *Readings in Information Retrieval*. Morgan Kaufman, Palo Alto, California.
- 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: *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.
- Turpin, A. H.; Hersh, W.** (2001). Why batch and user evaluations do not give the same results. In: Croft, W.; Harper, D.; Kraft, D.; Zobel, J. (Hrsg.): *Proceedings of the 24th Annual International Conference on Research and development in Information Retrieval*, S. 225–231. ACM Press, 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. D.** (1988). *Principles of Database and Knowledge-Base Systems*, Band I. Computer Science Press, Rockville (Md.).
- Ullman, J. D.** (1989). *Principles of Database and Knowledge-Base Systems: The New Technologies*, Band II. 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.
- Voorhees, E.; Harman, D. (Hrsg.)** (1998). *The Sixth Text REtrieval Conference (TREC-6)*, Gaithersburg, MD, USA. NIST.
- Voorhees, E.; Harman, D.** (2000). Overview of the Eighth Text REtrieval Conference (TREC-8). In: *The Eighth Text REtrieval Conference (TREC-8)*, S. 1–24. NIST, Gaithersburg, MD, USA.
- Westbrook, J.; Rumble, J. (Hrsg.)** (1983). *Computerized Materials Data Systems*, Gaithersburg, Maryland 20899, USA. National Bureau of Standards.
- Willet, P.** (1988). Recent Trends in Hierarchic Document Clustering: A Critical Review. *Information Processing and Management* 24(5), S. 577–597.
- 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.; Yao, Y.** (1995). On Modeling Information Retrieval with Probabilistic Inference. *ACM Transactions on Information Systems* 13(1), S. 38–68.
- 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: *Wissensbasierte Informationssysteme und Informationsmanagement*, S. 125–144. Universitätsverlag Konstanz.