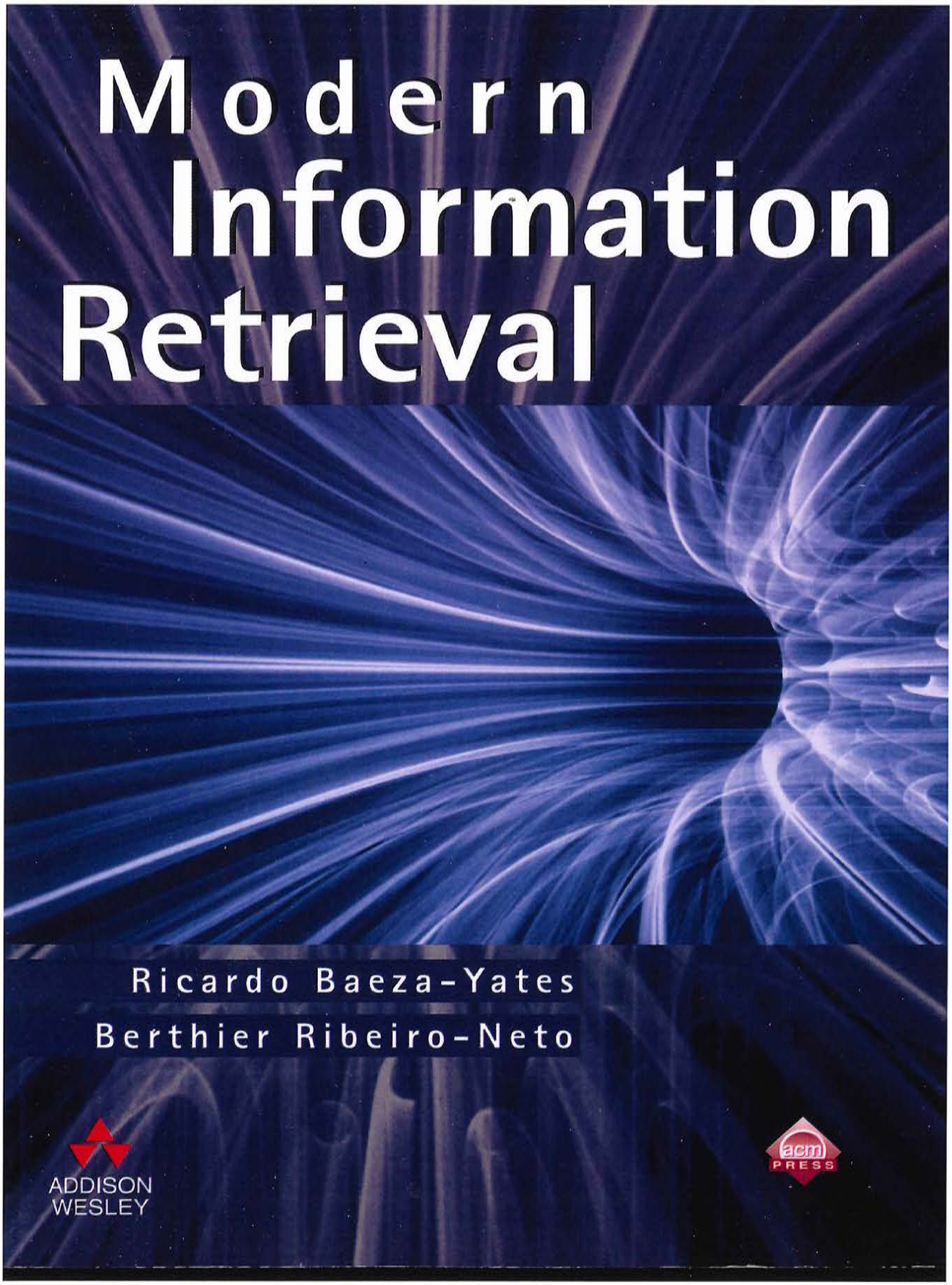
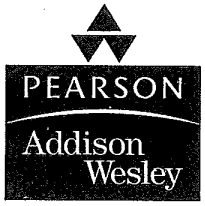


Modern Information Retrieval

Ricardo Baeza-Yates
Berthier Ribeiro-Neto

ACM Press
New York



Harlow, England • London • New York • Boston • San Francisco • Toronto • Sydney • Singapore • Hong Kong
Tokyo • Seoul • Taipei • New Delhi • Cape Town • Madrid • Mexico City • Amsterdam • Munich • Paris • Milan

Pearson Education Limited

Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsoned.co.uk

Typeset in Computer Modern by 56
Printed and bound in the United States of America

First printed 1999

ISBN 0-201-39829-X

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
Baeza-Yates, R. (Ricardo)
Modern information retrieval / Ricardo Baeza-Yates, Berthier Ribeiro-Neto.
p. cm.
Includes bibliographical reference and index.
ISBN 0-201-39829-X
1. Information storage and retrieval systems. I. Ribeiro, Berthier de Araûjo Neto, 1960-. II. Title.
Z667.B34 1999
025.04–dc21

99-10033
CIP

10 9
07 06 05 04

Preface

Information retrieval (IR) has changed considerably in recent years with the expansion of the World Wide Web and the advent of modern and inexpensive graphical user interfaces and mass storage devices. As a result, traditional IR textbooks have become quite out of date and this has led to the introduction of new IR books. Nevertheless, we believe that there is still great need for a book that approaches the field in a rigorous and complete way from a computer-science perspective (as opposed to a user-centered perspective). This book is an effort to partially fulfill this gap and should be useful for a first course on information retrieval as well as for a graduate course on the topic.

The book comprises two portions which complement and balance each other. The core portion includes nine chapters authored or coauthored by the designers of the book. The second portion, which is fully integrated with the first, is formed by six state-of-the-art chapters written by leading researchers in their fields. The same notation and glossary are used in all the chapters. Thus, despite the fact that several people have contributed to the text, this book is really much more a textbook than an edited collection of chapters written by separate authors. Furthermore, unlike a collection of chapters, we have carefully designed the contents and organization of the book to present a cohesive view of all the important aspects of modern information retrieval.

From IR models to indexing text, from IR visual tools and interfaces to the Web, from IR multimedia to digital libraries, the book provides both breadth of coverage and richness of detail. It is our hope that, given the now clear relevance and significance of information retrieval to modern society, the book will contribute to further disseminate the study of the discipline at information science, computer science, and library science departments throughout the world.

Ricardo Baeza-Yates, Santiago, Chile
Berthier Ribeiro-Neto, Belo Horizonte, Brazil
January, 1999

To Helena, Rosa, and our children

Amo los libros
exploradores,
libros con bosque o nieve,
profundidad o cielo

de Oda al Libro (I),

Pablo Neruda

I love books
that explore,
books with a forest or snow,
depth or sky

from Ode to the Book (I),

Pablo Neruda

território de homens livres
que será nosso país
e será pátria de todos.
Irmãos, cantai ese mundo
que não verei, mas virá
um dia, dentro de mil anos,
talvez mais... não tenho pressa.

de Cidade Prevista no livro
A Rosa do Povo, 1945

Carlos Drummond de Andrade

territory of free men
that will be our country
and will be the nation of all
Brothers, sing this world
which I'll not see, but which will come
one day, in a thousand years,
maybe more... no hurry.

from Prevised City in the book
The Rose of the People, 1945

Carlos Drummond de Andrade

Fifth, we thank the support of our institutions, the Departments of Computer Science of the University of Chile and of the Federal University of Minas Gerais, as well as the funding provided by national research agencies (CNPq in Brazil and CONICYT in Chile) and international collaboration projects, in particular CYTED project VII.13 AMYRI (Environment for Information Managing and Retrieval in the World Wide Web) and Finep project SIAM (Information Systems for Mobile Computers) under the Pronex program.

Most important, to Helena, Rosa, and our children, who put up with a string of trips abroad, lost weekends, and odd working hours.

List of Trademarks

Alta Vista is a trademark of Compaq Computer Corporation
FrameMaker is a trademark of Adobe Systems Incorporated
IBM SP2 is a trademark of International Business Machines Corporation
Netscape Communicator is a trademark of Netscape Communications Corporation
Solaris, Sun 3/50 and Sun UltraSparc-1 are trademarks of Sun Microsystems, Inc.
Thinking Machines CM-2 is a trademark of Thinking Machines Corporation
Unix is licensed through X/Open Company Ltd
Word is a trademark of Microsoft Corporation
WordPerfect is a trademark of of Corel Corporation

Contents

Chapter 1
Introduction

1.1 Motivation

Information retrieval (IR) deals with the representation, storage, organization of, and access to information items. The representation and organization of the information items should provide the user with easy access to the information in which he is interested. Unfortunately, characterization of the *user information need* is not a simple problem. Consider, for instance, the following hypothetical user information need in the context of the World Wide Web (or just the Web):

> Find all the pages (documents) containing information on college tennis teams which: (1) are maintained by an university in the USA and (2) participate in the NCAA tennis tournament. To be relevant, the page must include information on the national ranking of the team in the last three years and the email or phone number of the team coach.

Clearly, this full description of the user information need cannot be used directly to request information using the current interfaces of Web search engines. Instead, the user must first translate this information need into a *query* which can be processed by the search engine (or IR system).

In its most common form, this translation yields a set of keywords (or index terms) which summarizes the description of the user information need. Given the user query, the key goal of an IR system is to retrieve information which might be useful or relevant to the user. The emphasis is on the retrieval of *information* as opposed to the retrieval of *data*.

1.1.1 Information versus Data Retrieval

Data retrieval, in the context of an IR system, consists mainly of determining which documents of a collection contain the keywords in the user query which, most frequently, is not enough to satisfy the user information need. In fact, the user of an IR system is concerned more with retrieving *information* about a

subject than with retrieving data which satisfies a given query. A data retrieval language aims at retrieving all objects which satisfy clearly defined conditions such as those in a regular expression or in a relational algebra expression. Thus, for a data retrieval system, a single erroneous object among a thousand retrieved objects means total failure. For an information retrieval system, however, the retrieved objects might be inaccurate and small errors are likely to go unnoticed. The main reason for this difference is that information retrieval usually deals with natural language text which is not always well structured and could be semantically ambiguous. On the other hand, a data retrieval system (such as a relational database) deals with data that has a well defined structure and semantics.

Data retrieval, while providing a solution to the user of a database system, does not solve the problem of retrieving information about a subject or topic. To be effective in its attempt to satisfy the user information need, the IR system must somehow 'interpret' the contents of the information items (documents) in a collection and rank them according to a degree of relevance to the user query. This 'interpretation' of a document content involves extracting syntactic and semantic information from the document text and using this information to match the user information need. The difficulty is not only knowing how to extract this information but also knowing how to use it to decide relevance. Thus, the notion of *relevance* is at the center of information retrieval. In fact, the primary goal of an IR system is to retrieve all the documents which are relevant to a user query while retrieving as few non-relevant documents as possible.

1.1.2 Information Retrieval at the Center of the Stage

In the past 20 years, the area of information retrieval has grown well beyond its primary goals of indexing text and searching for useful documents in a collection. Nowadays, research in IR includes modeling, document classification and categorization, systems architecture, user interfaces, data visualization, filtering, languages, etc. Despite its maturity, until recently, IR was seen as a narrow area of interest mainly to librarians and information experts. Such a tendentious vision prevailed for many years, despite the rapid dissemination, among users of modern personal computers, of IR tools for multimedia and hypertext applications. In the beginning of the 1990s, a single fact changed once and for all these perceptions — the introduction of the World Wide Web.

The Web is becoming a universal repository of human knowledge and culture which has allowed unprecedent sharing of ideas and information in a scale never seen before. Its success is based on the conception of a standard user interface which is always the same no matter what computational environment is used to run the interface. As a result, the user is shielded from details of communication protocols, machine location, and operating systems. Further, any user can create his own Web documents and make them point to any other Web documents without restrictions. This is a key aspect because it turns the Web into a new publishing medium accessible to everybody. As an immediate

consequence, any Web user can push his personal agenda with little effort and almost at no cost. This universe without frontiers has attracted tremendous attention from millions of people everywhere since the very beginning. Furthermore, it is causing a revolution in the way people use computers and perform their daily tasks. For instance, home shopping and home banking are becoming very popular and have generated several hundred million dollars in revenues.

Despite so much success, the Web has introduced new problems of its own. Finding useful information on the Web is frequently a tedious and difficult task. For instance, to satisfy his information need, the user might navigate the space of Web links (i.e., the *hyperspace*) searching for information of interest. However, since the hyperspace is vast and almost unknown, such a navigation task is usually inefficient. For naive users, the problem becomes harder, which might entirely frustrate all their efforts. The main obstacle is the absence of a well defined underlying data model for the Web, which implies that information definition and structure is frequently of low quality. These difficulties have attracted renewed interest in IR and its techniques as promising solutions. As a result, almost overnight, IR has gained a place with other technologies at the center of the stage.

1.1.3 Focus of the Book

Despite the great increase in interest in information retrieval, modern textbooks on IR with a broad (and extensive) coverage of the various topics in the field are still difficult to find. In an attempt to partially fulfill this gap, this book presents an overall view of research in IR from a computer scientist's perspective. This means that the focus of the book is on computer algorithms and techniques used in information retrieval systems. A rather distinct viewpoint is taken by librarians and information science researchers, who adopt a human-centered interpretation of the IR problem. In this interpretation, the focus is on trying to understand how people interpret and use information as opposed to how to structure, store, and retrieve information automatically. While most of this book is dedicated to the computer scientist's viewpoint of the IR problem, the human-centered viewpoint is discussed to some extent in the last two chapters.

We put great emphasis on the integration of the different areas which are closed related to the information retrieval problem and thus, should be treated together. For that reason, besides covering text retrieval, library systems, user interfaces, and the Web, this book also discusses visualization, multimedia retrieval, and digital libraries.

1.2 Basic Concepts

The effective retrieval of relevant information is directly affected both by the *user task* and by the *logical view of the documents* adopted by the retrieval system, as we now discuss.

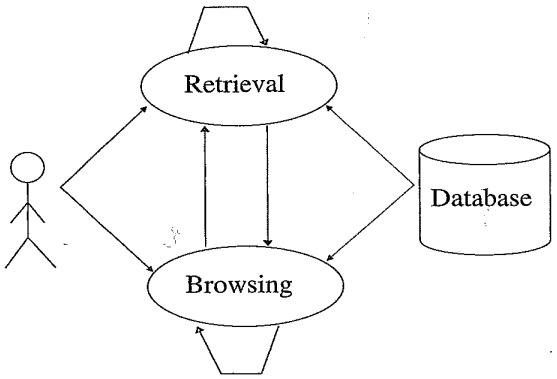


Figure 1.1 Interaction of the user with the retrieval system through distinct tasks.

1.2.1 The User Task

The user of a retrieval system has to translate his information need into a query in the language provided by the system. With an information retrieval system, this normally implies specifying a set of words which convey the semantics of the information need. With a data retrieval system, a query expression (such as, for instance, a regular expression) is used to convey the constraints that must be satisfied by objects in the answer set. In both cases, we say that the user searches for useful information executing a *retrieval* task.

Consider now a user who has an interest which is either poorly defined or which is inherently broad. For instance, the user might be interested in documents about car racing in general. In this situation, the user might use an interactive interface to simply look around in the collection for documents related to car racing. For instance, he might find interesting documents about Formula 1 racing, about car manufacturers, or about the '24 Hours of Le Mans.' Furthermore, while reading about the '24 Hours of Le Mans', he might turn his attention to a document which provides directions to Le Mans and, from there, to documents which cover tourism in France. In this situation, we say that the user is *browsing* the documents in the collection, not searching. It is still a process of retrieving information, but one whose main objectives are not clearly defined in the beginning and whose purpose might change during the interaction with the system.

In this book, we make a clear distinction between the different tasks the user of the retrieval system might be engaged in. His task might be of two distinct types: information or data *retrieval* and *browsing*. Classic information retrieval systems normally allow information or data retrieval. Hypertext systems are usually tuned for providing quick browsing. Modern digital library and Web interfaces might attempt to combine these tasks to provide improved retrieval capabilities. However, combination of retrieval and browsing is not yet a well

established approach and is not the dominant paradigm.

Figure 1.1 illustrates the interaction of the user through the different tasks we identify. Information and data retrieval are usually provided by most modern information retrieval systems (such as Web interfaces). Further, such systems might also provide some (still limited) form of browsing. While combining information and data retrieval with browsing is not yet a common practice, it might become so in the future.

Both retrieval and browsing are, in the language of the World Wide Web, 'pulling' actions. That is, the user requests the information in an interactive manner. An alternative is to do retrieval in an automatic and permanent fashion using software agents which *push* the information towards the user. For instance, information useful to a user could be extracted periodically from a news service. In this case, we say that the IR system is executing a particular retrieval task which consists of *filtering* relevant information for later inspection by the user. We briefly discuss filtering in Chapter 2.

1.2.2 Logical View of the Documents

Due to historical reasons, documents in a collection are frequently represented through a set of index terms or keywords. Such keywords might be extracted directly from the text of the document or might be specified by a human subject (as frequently done in the information sciences arena). No matter whether these representative keywords are derived automatically or generated by a specialist, they provide a *logical view of the document*. For a precise definition of the concept of a document and its characteristics, see Chapter 6.

Modern computers are making it possible to represent a document by its full set of words. In this case, we say that the retrieval system adopts a *full text* logical view (or representation) of the documents. With very large collections, however, even modern computers might have to reduce the set of representative keywords. This can be accomplished through the elimination of *stopwords* (such as articles and connectives), the use of *stemming* (which reduces distinct words to their common grammatical root), and the identification of noun groups (which eliminates adjectives, adverbs, and verbs). Further, compression might be employed. These operations are called *text operations* (or transformations) and are covered in detail in Chapter 7. Text operations reduce the complexity of the document representation and allow moving the logical view from that of a full text to that of a set of *index terms*.

The full text is clearly the most complete logical view of a document but its usage usually implies higher computational costs. A small set of categories (generated by a human specialist) provides the most concise logical view of a document but its usage might lead to retrieval of poor quality. Several intermediate logical views (of a document) might be adopted by an information retrieval system as illustrated in Figure 1.2. Besides adopting any of the intermediate representations, the retrieval system might also recognize the internal structure normally present in a document (e.g., chapters, sections, subsections, etc.). This

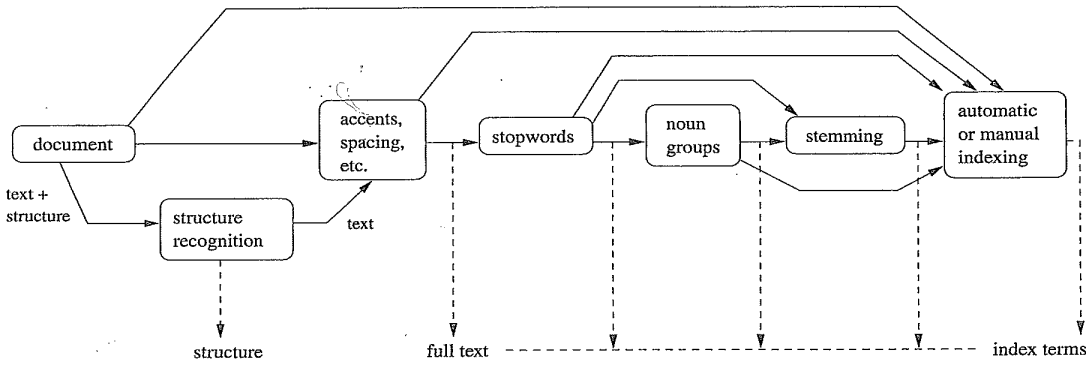


Figure 1.2 Logical view of a document: from full text to a set of index terms.

information on the structure of the document might be quite useful and is required by structured text retrieval models such as those discussed in Chapter 2.

As illustrated in Figure 1.2, we view the issue of logically representing a document as a continuum in which the logical view of a document might shift (smoothly) from a full text representation to a higher level representation specified by a human subject.

1.3 Past, Present, and Future

1.3.1 Early Developments

For approximately 4000 years, man has organized information for later retrieval and usage. A typical example is the table of contents of a book. Since the volume of information eventually grew beyond a few books, it became necessary to build specialized data structures to ensure faster access to the stored information. An old and popular data structure for faster information retrieval is a collection of selected words or concepts with which are associated pointers to the related information (or documents) — the *index*. In one form or another, indexes are at the core of every modern information retrieval system. They provide faster access to the data and allow the query processing task to be speeded up. A detailed coverage of indexes and their usage for searching can be found in Chapter 8.

For centuries, indexes were created manually as categorization hierarchies. In fact, most libraries still use some form of categorical hierarchy to classify their volumes (or documents), as discussed in Chapter 14. Such hierarchies have usually been conceived by human subjects from the library sciences field. More recently, the advent of modern computers has made possible the construction of large indexes automatically. Automatic indexes provide a view of the retrieval problem which is much more related to the system itself than to the user need.

In this respect, it is important to distinguish between two different views of the IR problem: a computer-centered one and a human-centered one.

In the computer-centered view, the IR problem consists mainly of building up efficient indexes, processing user queries with high performance, and developing ranking algorithms which improve the 'quality' of the answer set. In the human-centered view, the IR problem consists mainly of studying the behavior of the user, of understanding his main needs, and of determining how such understanding affects the organization and operation of the retrieval system. According to this view, keyword based query processing might be seen as a strategy which is unlikely to yield a good solution to the information retrieval problem in the long run.

In this book, we focus mainly on the computer-centered view of the IR problem because it continues to be dominant in the market place.

1.3.2 Information Retrieval in the Library

Libraries were among the first institutions to adopt IR systems for retrieving information. Usually, systems to be used in libraries were initially developed by academic institutions and later by commercial vendors. In the first generation, such systems consisted basically of an automation of previous technologies (such as card catalogs) and basically allowed searches based on author name and title. In the second generation, increased search functionality was added which allowed searching by subject headings, by keywords, and some more complex query facilities. In the third generation, which is currently being deployed, the focus is on improved graphical interfaces, electronic forms, hypertext features, and open system architectures.

Traditional library management system vendors include Endeavor Information Systems Inc., Innovative Interfaces Inc., and EOS International. Among systems developed with a research focus and used in academic libraries, we distinguish Okapi (at City University, London), MELVYL (at University of California), and Cheshire II (at UC Berkeley). Further details on these library systems can be found in Chapter 14.

1.3.3 The Web and Digital Libraries

If we consider the search engines on the Web today, we conclude that they continue to use indexes which are very similar to those used by librarians a century ago. What has changed then?

Three dramatic and fundamental changes have occurred due to the advances in modern computer technology and the boom of the Web. First, it became a lot cheaper to have access to various sources of information. This allows reaching a wider audience than ever possible before. Second, the advances in all kinds of digital communication provided greater access to networks. This implies that the information source is available even if distantly located and that

the access can be done quickly (frequently, in a few seconds). Third, the freedom to post whatever information someone judges useful has greatly contributed to the popularity of the Web. For the first time in history, many people have free access to a large publishing medium.

Fundamentally, low cost, greater access, and publishing freedom have allowed people to use the Web (and modern digital libraries) as a highly *interactive* medium. Such interactivity allows people to exchange messages, photos, documents, software, videos, and to 'chat' in a convenient and low cost fashion. Further, people can do it at the time of their preference (for instance, you can buy a book late at night) which further improves the convenience of the service. Thus, high interactivity is the fundamental and current shift in the communication paradigm. Searching the Web is covered in Chapter 13, while digital libraries are covered in Chapter 15.

In the future, three main questions need to be addressed. First, despite the high interactivity, people still find it difficult (if not impossible) to retrieve information relevant to their information needs. Thus, in the dynamic world of the Web and of large digital libraries, which techniques will allow retrieval of higher quality? Second, with the ever increasing demand for access, quick response is becoming more and more a pressing factor. Thus, which techniques will yield faster indexes and smaller query response times? Third, the quality of the retrieval task is greatly affected by the user interaction with the system. Thus, how will a better understanding of the user behavior affect the design and deployment of new information retrieval strategies?

1.3.4 Practical Issues

Electronic commerce is a major trend on the Web nowadays and one which has benefited millions of people. In an electronic transaction, the buyer usually has to submit to the vendor some form of credit information which can be used for charging for the product or service. In its most common form, such information consists of a credit card number. However, since transmitting credit card numbers over the Internet is not a safe procedure, such data is usually transmitted over a fax line. This implies that, at least in the beginning, the transaction between a new user and a vendor requires executing an off-line procedure of several steps before the actual transaction can take place. This situation can be improved if the data is encrypted for security. In fact, some institutions and companies already provide some form of encryption or automatic authentication for security reasons.

However, security is not the only concern. Another issue of major interest is privacy. Frequently, people are willing to exchange information as long as it does not become public. The reasons are many but the most common one is to protect oneself against misuse of private information by third parties. Thus, privacy is another issue which affects the deployment of the Web and which has not been properly addressed yet.

Two other very important issues are copyright and patent rights. It is far

from clear how the wide spread of data on the Web affects copyright and patent laws in the various countries. This is important because it affects the business of building up and deploying large digital libraries. For instance, is a site which supervises all the information it posts acting as a publisher? And if so, is it responsible for a misuse of the information it posts (even if it is not the source)?

Additionally, other practical issues of interest include scanning, optical character recognition (OCR), and cross-language retrieval (in which the query is in one language but the documents retrieved are in another language). In this book, however, we do not cover practical issues in detail because it is not our main focus. The reader interested in details of practical issues is referred to the interesting book by Lesk [501].

1.4 The Retrieval Process

At this point, we are ready to detail our view of the retrieval process. Such a process is interpreted in terms of component subprocesses whose study yields many of the chapters in this book.

To describe the retrieval process, we use a simple and generic software architecture as shown in Figure 1.3. First of all, before the retrieval process can even be initiated, it is necessary to define the text database. This is usually done by the manager of the database, which specifies the following: (a) the documents to be used, (b) the operations to be performed on the text, and (c) the text model (i.e., the text structure and what elements can be retrieved). The text operations transform the original documents and generate a logical view of them.

Once the logical view of the documents is defined, the database manager (using the DB Manager Module) builds an index of the text. An index is a critical data structure because it allows fast searching over large volumes of data. Different index structures might be used, but the most popular one is the *inverted file* as indicated in Figure 1.3. The resources (time and storage space) spent on defining the text database and building the index are amortized by querying the retrieval system many times.

Given that the document database is indexed, the retrieval process can be initiated. The user first specifies a *user need* which is then parsed and transformed by the same text operations applied to the text. Then, *query operations* might be applied before the actual *query*, which provides a system representation for the user need, is generated. The query is then processed to obtain the *retrieved documents*. Fast query processing is made possible by the index structure previously built.

Before been sent to the user, the retrieved documents are ranked according to a *likelihood* of relevance. The user then examines the set of ranked documents in the search for useful information. At this point, he might pinpoint a subset of the documents seen as definitely of interest and initiate a *user feedback* cycle. In such a cycle, the system uses the documents selected by the user to change the query formulation. Hopefully, this modified query is a better representation

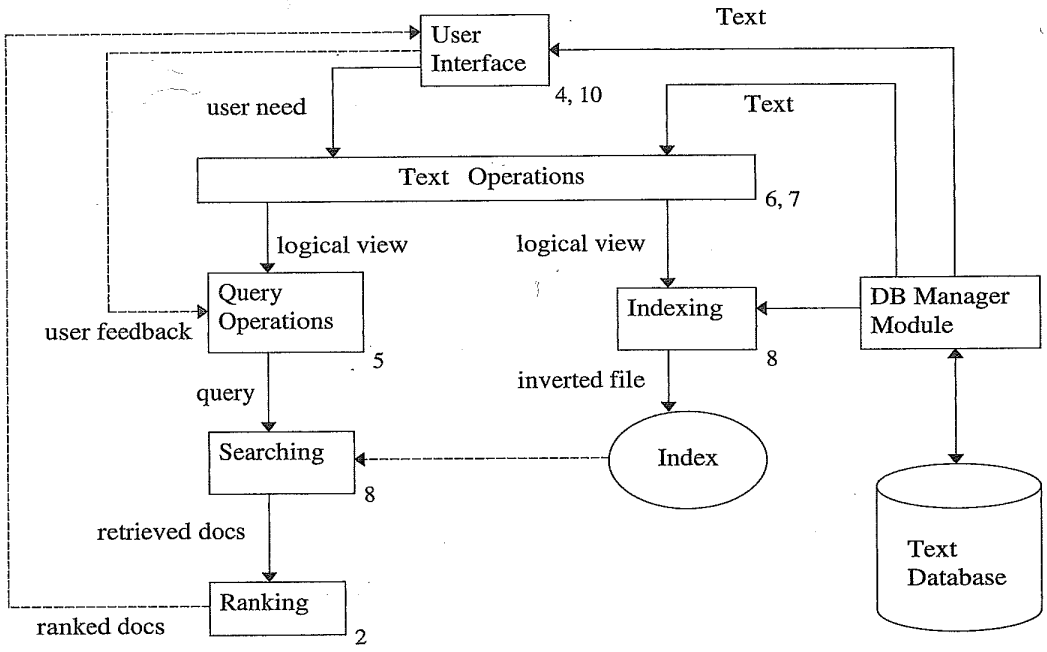


Figure 1.3 The process of retrieving information (the numbers beside each box indicate the chapters that cover the corresponding topic).

of the real user need.

The small numbers outside the lower right corner of various boxes in Figure 1.3 indicate the chapters in this book which discuss the respective subprocesses in detail. A brief introduction to each of these chapters can be found in section 1.5.

Consider now the user interfaces available with current information retrieval systems (including Web search engines and Web browsers). We first notice that the user almost never declares his information need. Instead, he is required to provide a direct representation for the query that the system will execute. Since most users have no knowledge of text and query operations, the query they provide is frequently inadequate. Therefore, it is not surprising to observe that poorly formulated queries lead to poor retrieval (as happens so often on the Web).

1.5 Organization of the Book

For ease of comprehension, this book has a straightforward structure in which four main parts are distinguished: text IR, human-computer interaction (HCI)

for IR, multimedia IR, and applications of IR. *Text IR* discusses the classic problem of searching a collection of documents for useful information. *HCI for IR* discusses current trends in IR towards improved user interfaces and better data visualization tools. *Multimedia IR* discusses how to index document images and other binary data by extracting features from their content and how to search them efficiently. On the other hand, document images that are predominantly text (rather than pictures) are called *textual images* and are amenable to automatic extraction of keywords through metadescriptors, and can be retrieved using text IR techniques. *Applications of IR* covers modern applications of IR such as the Web, bibliographic systems, and digital libraries. Each part is divided into topics which we now discuss.

1.5.1 Book Topics

The four parts which compose this book are subdivided into eight topics as illustrated in Figure 1.4. These eight topics are as follows.

The topic *Retrieval Models & Evaluation* discusses the traditional models of searching text for useful information and the procedures for evaluating an information retrieval system. The topic *Improvements on Retrieval* discusses techniques for transforming the query and the text of the documents with the aim of improving retrieval. The topic *Efficient Processing* discusses indexing and searching approaches for speeding up the retrieval. These three topics compose the first part on Text IR.

The topic *Interfaces & Visualization* covers the interaction of the user with the information retrieval system. The focus is on interfaces which facilitate the process of specifying a query and provide a good visualization of the results.

The topic *Multimedia Modeling & Searching* discusses the utilization of multimedia data with information retrieval systems. The focus is on modeling, indexing, and searching multimedia data such as voice, images, and other binary data.

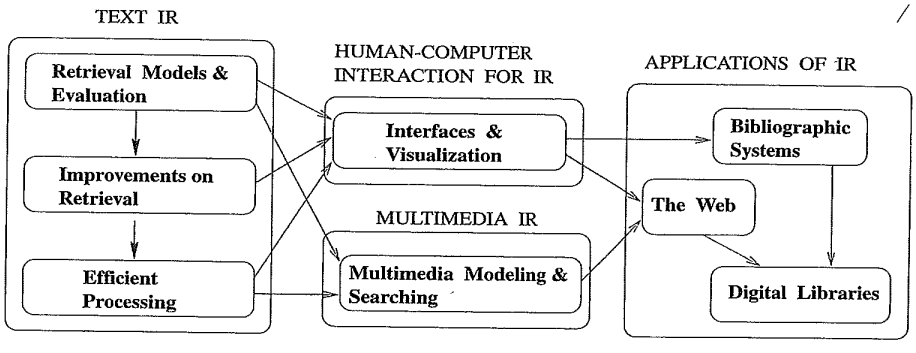


Figure 1.4 Topics which compose the book and their relationships.

The part on applications of IR is composed of three interrelated topics: *The Web*, *Bibliographic Systems*, and *Digital Libraries*. Techniques developed for the first two applications support the deployment of the latter.

The eight topics distinguished above generate the 14 chapters, besides this introduction, which compose this book and which we now briefly introduce.

1.5.2 Book Chapters

Figure 1.5 illustrates the overall structure of this book. The reasoning which yielded the chapters from 2 to 15 is as follows.

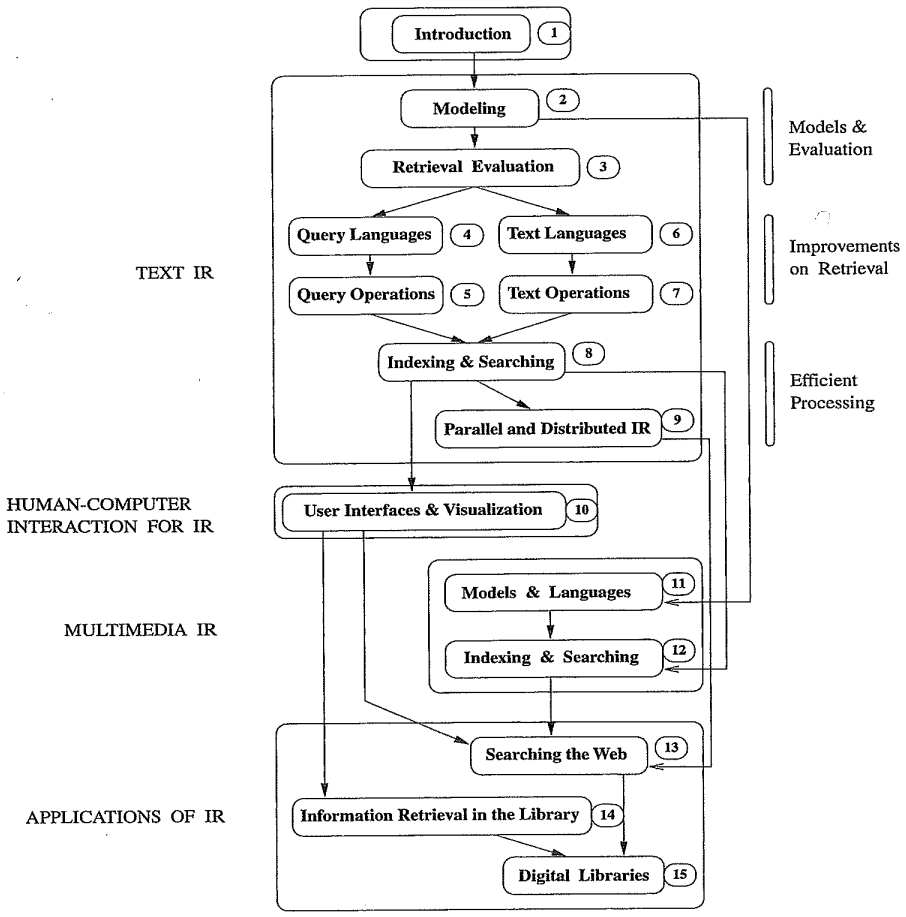


Figure 1.5 Structure of the book.

In the traditional keyword-based approach, the user specifies his information need by providing sets of keywords and the information system retrieves the documents which best approximate the user query. Also, the information system

might attempt to rank the retrieved documents using some measure of relevance. This ranking task is critical in the process of attempting to satisfy the user information need and is the main goal of *modeling* in IR. Thus, information retrieval models are discussed early in Chapter 2. The discussion introduces many of the fundamental concepts in information retrieval and lays down much of the foundation for the subsequent chapters. Our coverage is detailed and broad. Classic models (Boolean, vector, and probabilistic), modern probabilistic variants (belief network models), alternative paradigms (extended Boolean, generalized vector, latent semantic indexing, neural networks, and fuzzy retrieval), structured text retrieval, and models for browsing (hypertext) are all carefully introduced and explained.

Once a new retrieval algorithm (maybe based on a new retrieval model) is conceived, it is necessary to evaluate its performance. Traditional evaluation strategies usually attempt to estimate the costs of the new algorithm in terms of time and space. With an information retrieval system, however, there is the additional issue of evaluating the relevance of the documents retrieved. For this purpose, text reference collections and evaluation procedures based on variables other than time and space are used. Chapter 3 is dedicated to the discussion of *retrieval evaluation.*

In traditional IR, queries are normally expressed as a set of keywords which is quite convenient because the approach is simple and easy to implement. However, the simplicity of the approach prevents the formulation of more elaborate querying tasks. For instance, queries which refer to both the structure and the content of the text cannot be formulated. To overcome this deficiency, more sophisticated query languages are required. Chapter 4 discusses various types of *query languages.* Since now the user might refer to the structure of a document in his query, this structure has to be defined. This is done by embedding the description of a document content and of its structure in a text language such as the Standard Generalized Markup Language (SGML). As illustrated in Figure 1.5, Chapter 6 is dedicated to the discussion of *text languages.*

Retrieval based on keywords might be of fairly low quality. Two possible reasons are as follows. First, the user query might be composed of too few terms which usually implies that the query context is poorly characterized. This is frequently the case, for instance, in the Web. This problem is dealt with through transformations in the query such as query expansion and user relevance feedback. Such *query operations* are covered in Chapter 5. Second, the set of keywords generated for a given document might fail to summarize its semantic content properly. This problem is dealt with through transformations in the text such as identification of noun groups to be used as keywords, stemming, and the use of a thesaurus. Additionally, for reasons of efficiency, text compression can be employed. Chapter 7 is dedicated to *text operations.*

Given the user query, the information system has to retrieve the documents which are related to that query. The potentially large size of the document collection (e.g., the Web is composed of millions of documents) implies that specialized indexing techniques must be used if efficient retrieval is to be achieved. Thus, to speed up the task of matching documents to queries, proper *indexing* and *search-*

ing techniques are used as discussed in Chapter 8. Additionally, query processing can be further accelerated through the adoption of *parallel and distributed IR* techniques as discussed in Chapter 9.

As illustrated in Figure 1.5, all the key issues regarding Text IR, from modeling to fast query processing, are covered in this book.

Modern user interfaces implement strategies which assist the user to form a query. The main objective is to allow him to define more precisely the context associated to his information need. The importance of query contextualization is a consequence of the difficulty normally faced by users during the querying process. Consider, for instance, the problem of quickly finding useful information in the Web. Navigation in hyperspace is not a good solution due to the absence of a logical and semantically well defined structure (the Web has no underlying logical model). A popular approach for specifying a user query in the Web consists of providing a set of keywords which are searched for. Unfortunately, the number of terms provided by a common user is small (typically, fewer than four) which usually implies that the query is vague. This means that new user interface paradigms which assist the user with the query formation process are required. Further, since a vague user query usually retrieves hundreds of documents, the conventional approach of displaying these documents as items of a scrolling list is clearly inadequate. To deal with this problem, new data visualization paradigms have been proposed in recent years. The main trend is towards visualization of a large subset of the retrieved documents at once and direct manipulation of the whole subset. *User interfaces* for assisting the user to form his query and current approaches for *visualization* of large data sets are covered in Chapter 10.

Following this, we discuss the application of IR techniques to multimedia data. The key issue is how to model, index, and search structured documents which contain multimedia objects such as digitized voice, images, and other binary data. *Models and query languages* for office and medical information retrieval systems are covered in Chapter 11. *Efficient indexing and searching* of multimedia objects is covered in Chapter 12. Some readers may argue that the models and techniques for multimedia retrieval are rather different from those for classic text retrieval. However, we take into account that images and text are usually together and that with the Web, other media types (such as video and audio) can also be mixed in. Therefore, we believe that in the future, all the above will be treated in a unified and consistent manner. Our book is a first step in that direction.

The final three chapters of the book are dedicated to applications of modern information retrieval: the Web, bibliographic systems, and digital libraries. As illustrated in Figure 1.5, Chapter 13 presents the Web and discusses the main problems related to the issue of searching the Web for useful information. Also, our discussion covers briefly the most popular search engines in the Web presenting particularities of their organization. Chapter 14 covers commercial document databases and online public access catalogs. Commercial document databases are still the largest information retrieval systems nowadays. LEXIS-NEXIS, for instance, has a database with 1.3 billion documents and attends to over 120 million query requests annually. Finally, Chapter 15 discusses modern digital

libraries. Architectural issues, models, prototypes, and standards are all covered. The discussion also introduces the '5S' model (streams, structures, spaces, scenarios and societies) as a framework for providing theoretical and practical unification of digital libraries.

1.6 How to Use this Book

Although several people have contributed chapters for this book, it is really a textbook. The contents and the structure of the book have been carefully designed by the two main authors who also authored or coauthored nine of the 15 chapters in the book. Further, all the contributed chapters have been judiciously edited and integrated into a unifying framework that provides uniformity in structure and style, a common glossary, a common bibliography, and appropriate cross-references. At the end of each chapter, a discussion on research issues, trends, and selected bibliography is included. This discussion should be useful for graduate students as well as for researchers. Furthermore, the book is complemented by a Web page with additional information and resources.

1.6.1 Teaching Suggestions

This textbook can be used in many different areas including computer science (CS), information systems, and library science. The following list gives suggested contents for different courses at the undergraduate and graduate level, based on syllabuses of many universities around the world:

- **Information Retrieval** (Computer Science, undergraduate): this is the standard course for many CS programs. The minimum content should include Chapters 1 to 8 and Chapter 10, that is, most of the part on Text IR complemented with the chapter on user interfaces. Some specific topics of those chapters, such as more advanced models for IR and sophisticated algorithms for indexing and searching, can be omitted to fit a one term course. The chapters on Applications of IR can be mentioned briefly at the end.
- **Advanced Information Retrieval** (Computer Science, graduate): similar to the previous course but with more detailed coverage of the various chapters particularly modeling and searching (assuming the previous course as a requirement). In addition, Chapter 9 and Chapters 13 to 15 should be covered completely. Emphasis on research problems and new results is a must.
- **Information Retrieval** (Information Systems, undergraduate): this course is similar to the CS course, but with a different emphasis. It should include Chapters 1 to 7 and Chapter 10. Some notions from Chapter 8 are

useful but not crucial. At the end, the system-oriented parts of the chapters on Applications of IR, in particular those on Bibliographic Systems and Digital Libraries, must be covered (this material can be complemented with topics from [501]).

- **Information Retrieval** (Library Science, undergraduate): similar to the previous course, but removing the more technical and advanced material of Chapters 2, 5, and 7. Also, greater emphasis should be put on the chapters on Bibliographic Systems and Digital Libraries. The course should be complemented with a thorough discussion of the user-centered view of the IR problem (for example, using the book by Allen [13]).
- **Multimedia Retrieval** (Computer Science, undergraduate or graduate): this course should include Chapters 1 to 3, 6, and 11 to 15. The emphasis could be on multimedia itself or on the integration of classical IR with multimedia. The course can be complemented with one of the many books on this topic, which are usually more broad and technical.
- **Topics in IR** (Computer Science, graduate): many chapters of the book can be used for this course. It can emphasize modeling and evaluation or user interfaces and visualization. It can also be focused on algorithms and data structures (in that case, [275] and [825] are good complements). A multimedia focus is also possible, starting with Chapters 11 and 12 and using more specific books later on.
- **Topics in IR** (Information Systems or Library Science, graduate) similar to the above but with emphasis on non-technical parts. For example, the course could cover modeling and evaluation, query languages, user interfaces, and visualization. The chapters on applications can also be considered.
- **Web Retrieval and Information Access** (generic, undergraduate or graduate): this course should emphasize hypertext, concepts coming from networks and distributed systems and multimedia. The kernel should be the basic models of Chapter 2 followed by Chapters 3, 4, and 6. Also, Chapters 11 and 13 to 15 should be discussed.
- **Digital Libraries** (generic, undergraduate or graduate): This course could start with part of Chapters 2 to 4 and 6, followed by Chapters 10, 14, and 15. The kernel of the course could be based on the book by Lesk [501].

More bibliography useful for many of the courses above is discussed in the last section of this chapter.

1.6.2 The Book's Web Page

As IR is a very dynamic area nowadays, a book by itself is not enough. For that reason (and many others), the book has a Web home page located and mirrored in the following places (mirrors in USA and Europe are also planned):

- Brazil: `http://www.dcc.ufmg.br/irbook`
- Chile: `http://sunsite.dcc.uchile.cl/irbook`

Comments, suggestions, contributions, or mistakes found are welcome through email to the contact authors given on the Web page.

The Web page contains the Table of Contents, Preface, Acknowledgements, Introduction, Glossary, and other appendices to the book. It also includes exercises and teaching materials that will be increasing in volume and changing with time. In addition, a reference collection (containing 1239 documents on Cystic Fibrosis and 100 information requests with extensive relevance evaluation [721]) is available for experimental purposes. Furthermore, the page includes useful pointers to IR syllabuses in different universities, IR research groups, IR publications, and other resources related to IR and this book. Finally, any new important results or additions to the book as well as an errata will be made publicly available there.

1.7 Bibliographic Discussion

Many other books have been written on information retrieval, and due to the current widespread interest in the subject, new books have appeared recently. In the following, we briefly compare our book with these previously published works.

Classic references in the field of information retrieval are the books by van Rijsbergen [785] and Salton and McGill [698]. Our distinction between data and information retrieval is borrowed from the former. Our definition of the information retrieval process is influenced by the latter. However, almost 20 years later, both books are now outdated and do not cover many of the new developments in information retrieval.

Three more recent and also well known references in information retrieval are the book edited by Frakes and Baeza-Yates [275], the book by Witten, Moffat, and Bell [825], and the book by Lesk [501]. All these three books are complementary to this book. The first is focused on data structures and algorithms for information retrieval and is useful whenever quick prototyping of a known algorithm is desired. The second is focused on indexing and compression, and covers images besides text. For instance, our definition of a textual image is borrowed from it. The third is focused on digital libraries and practical issues such as history, distribution, usability, economics, and property rights. On the issue of computer-centered and user-centered retrieval, a generic book on information systems that takes the latter view is due to Allen [13].

There are other complementary books for specific chapters. For example, there are many books on IR and hypertext. The same is true for generic or specific multimedia retrieval, as images, audio or video. Although not an information retrieval title, the book by Rosenfeld and Morville [682] on information architecture of the Web, is a good complement to our chapter on searching the

Web. The book by Menasce and Almeida [554] demonstrates how to use queueing theory for predicting Web server performance. In addition, there are many books that explain how to find information on the Web and how to use search engines.

The reference edited by Sparck Jones and Willet [414], which was long awaited, is really a collection of papers rather than an edited book. The coherence and breadth of coverage in our book makes it more appropriate as a textbook in a formal discipline. Nevertheless, this collection is a valuable research tool. A collection of papers on cross-language information retrieval was recently edited by Grefenstette [323]. This book is a good complement to ours for people interested in this particular topic. Additionally, a collection focused on intelligent IR was edited recently by Maybury [550], and another collection on natural language IR edited by Strzalkowski will appear soon [748].

The book by Korfhage [451] covers a lot less material and its coverage is not as detailed as ours. For instance, it includes no detailed discussion of digital libraries, the Web, multimedia, or parallel processing. Similarly, the books by Kowalski [459] and Shapiro *et al.* [719] do not cover these topics in detail, and have a different orientation. Finally, the recent book by Grossman and Frieder [326] does not discuss the Web, digital libraries, or visual interfaces.

For people interested in research results, the main journals on IR are: *Journal of the American Society of Information Sciences* (*JASIS*) published by Wiley and Sons, *ACM Transactions on Information Systems, Information Processing & Management* (IP&M, Elsevier), *Information Systems* (Elsevier), *Information Retrieval* (Kluwer), and *Knowledge and Information Systems* (Springer). The main conferences are: ACM SIGIR International Conference on Information Retrieval, ACM International Conference on Digital Libraries (ACM DL), ACM Conference on Information Knowledge and Management (CIKM), and Text REtrieval Conference (TREC). Regarding events of regional influence, we would like to acknowledge the SPIRE (South American Symposium on String Processing and Information Retrieval) symposium.

Chapter 2
Modeling

2.1 Introduction

Traditional information retrieval systems usually adopt index terms to index and retrieve documents. In a restricted sense, an index term is a keyword (or group of related words) which has some meaning of its own (i.e., which usually has the semantics of a noun). In its more general form, an index term is simply any word which appears in the text of a document in the collection. Retrieval based on index terms is simple but raises key questions regarding the information retrieval task. For instance, retrieval using index terms adopts as a fundamental foundation the idea that the semantics of the documents and of the user information need can be naturally expressed through sets of index terms. Clearly, this is a considerable oversimplification of the problem because a lot of the semantics in a document or user request is lost when we replace its text with a set of words. Furthermore, matching between each document and the user request is attempted in this very imprecise space of index terms. Thus, it is no surprise that the documents retrieved in response to a user request expressed as a set of keywords are frequently irrelevant. If one also considers that most users have no training in properly forming their queries, the problem is worsened with potentially disastrous results. The frequent dissatisfaction of Web users with the answers they normally obtain is just one good example of this fact.

Clearly, one central problem regarding information retrieval systems is the issue of predicting which documents are relevant and which are not. Such a decision is usually dependent on a ranking algorithm which attempts to establish a simple ordering of the documents retrieved. Documents appearing at the top of this ordering are considered to be more likely to be relevant. Thus, ranking algorithms are at the core of information retrieval systems.

A ranking algorithm operates according to basic premises regarding the notion of document relevance. Distinct sets of premises (regarding document relevance) yield distinct information retrieval models. The IR model adopted determines the predictions of what is relevant and what is not (i.e., the notion of relevance implemented by the system). The purpose of this chapter is to cover the most important information retrieval models proposed over the years. By

doing so, the chapter also provides a conceptual basis for most of the remaining chapters in this book.

We first propose a taxonomy for categorizing the 15 IR models we cover. Second, we distinguish between two types of user retrieval tasks: ad hoc and filtering. Third, we present a formal characterization of IR models which is useful for distinguishing the various components of a particular model. Last, we discuss each of the IR models included in our taxonomy.

2.2 A Taxonomy of Information Retrieval Models

The three classic models in information retrieval are called Boolean, vector, and probabilistic. In the Boolean model, documents and queries are represented as sets of index terms. Thus, as suggested in [327], we say that the model is *set theoretic*. In the vector model, documents and queries are represented as vectors in a t-dimensional space. Thus, we say that the model is *algebraic*. In the probabilistic model, the framework for modeling document and query representations is based on probability theory. Thus, as the name indicates, we say that the model is *probabilistic*.

Over the years, alternative modeling paradigms for each type of classic model (i.e., set theoretic, algebraic, and probabilistic) have been proposed. Regarding alternative set theoretic models, we distinguish the fuzzy and the extended Boolean models. Regarding alternative algebraic models, we distinguish the generalized vector, the latent semantic indexing, and the neural network models. Regarding alternative probabilistic models, we distinguish the inference network and the belief network models. Figure 2.1 illustrates a taxonomy of these information retrieval models.

Besides references to the text content, the model might also allow references to the structure normally present in written text. In this case, we say that we have a structured model. We distinguish two models for structured text retrieval namely, the non-overlapping lists model and the proximal nodes model.

As discussed in Chapter 1, the user task might be one of *browsing* (instead of retrieval). In Figure 2.1, we distinguish three models for browsing namely, flat, structure guided, and hypertext.

The organization of this chapter follows the taxonomy of information retrieval models depicted in the figure.We first discuss the three classic models. Second, we discuss the alternative models for each type of classic model. Third, we cover structured text retrieval models. At the end, we discuss models for browsing.

We emphasize that the IR model (Boolean, vector, probabilistic, etc.), the logical view of the documents (full text, set of index terms, etc.), and the user task (retrieval, browsing) are orthogonal aspects of a retrieval system as detailed in Chapter 1. Thus, despite the fact that some models are more appropriate for a certain user task than for another, the same IR model can be used with distinct document logical views to perform different user tasks. Figure 2.2 illustrates the retrieval models most frequently associated with each one of six distinct combinations of a document logical view and a user task.

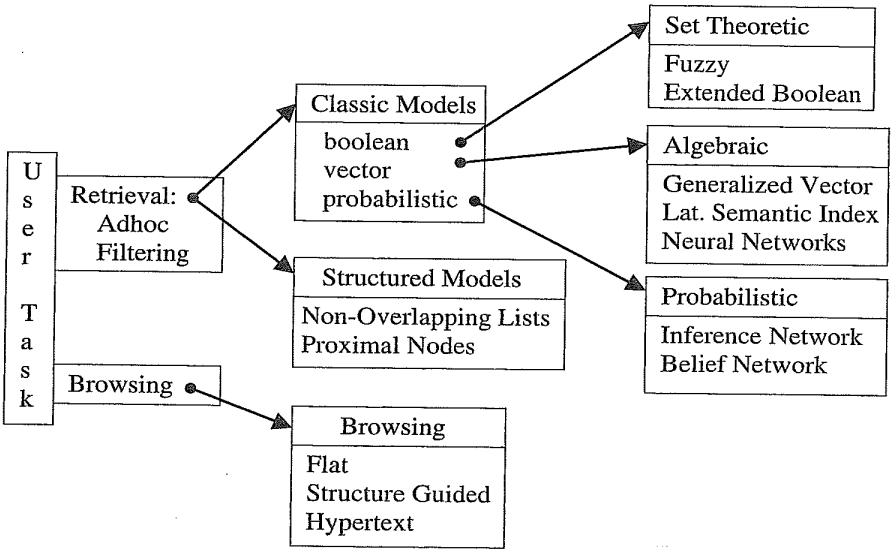


Figure 2.1 A taxonomy of information retrieval models.

USER TASK	LOGICAL VIEW OF DOCUMENTS: Index Terms	Full Text	Full Text + Structure
Retrieval	Classic Set Theoretic Algebraic Probabilistic	Classic Set Theoretic Algebraic Probabilistic	Structured
Browsing	Flat	Flat Hypertext	Structure Guided Hypertext

Figure 2.2 Retrieval models most frequently associated with distinct combinations of a document logical view and a user task.

2.3 Retrieval: Ad hoc and Filtering

In a conventional information retrieval system, the documents in the collection remain relatively static while new queries are submitted to the system. This operational mode has been termed *ad hoc* retrieval in recent years and is the

most common form of user task. A similar but distinct task is one in which the queries remain relatively static while new documents come into the system (and leave). For instance, this is the case with the stock market and with news wiring services. This operational mode has been termed *filtering*.

In a filtering task [74], a *user profile* describing the user's preferences is constructed. Such a profile is then compared to the incoming documents in an attempt to determine those which might be of interest to this particular user. For instance, this approach can be used to select a news article among thousands of articles which are broadcast each day. Other potential scenarios for the application of filtering include the selection of preferred judicial decisions, or the selection of articles from daily newspapers, etc.

Typically, the filtering task simply indicates to the user the documents which might be of interest to him. The task of determining which ones are really relevant is fully reserved to the user. Not even a ranking of the filtered documents is provided. A variation of this procedure is to rank the filtered documents and show this ranking to the user. The motivation is that the user can examine a smaller number of documents if he assumes that the ones at the top of this ranking are more likely to be relevant. This variation of filtering is called *routing* (see Chapter 3) but it is not popular.

Even if no ranking is presented to the user, the filtering task can compute an internal ranking to determine potentially relevant documents. For instance, documents with a ranking above a given threshold could be selected; the others would be discarded. Any IR model can be adopted to rank the documents, but the vector model is usually preferred due to its simplicity. At this point, we observe that filtering is really a type of user task (or operational mode) and not a model of information retrieval. Thus, the task of filtering and the IR model adopted are orthogonal aspects of an IR system.

In a filtering task, the crucial step is not the ranking itself but the construction of a user profile which truly reflects the user's preferences. Many approaches for constructing user profiles have been proposed and here we briefly discuss a couple of them.

A simplistic approach for constructing a user profile is to describe the profile through a set of keywords and to require the user to provide the necessary keywords. The approach is simplistic because it requires the user to do too much. In fact, if the user is not familiar with the service which generates the upcoming documents, he might find it fairly difficult to provide the keywords which appropriately describe his preferences in that context. Furthermore, an attempt by the user to familiarize himself with the vocabulary of the upcoming documents might turn into a tedious and time consuming exercise. Thus, despite its feasibility, requiring the user to precisely describe his profile might be impractical. A more elaborate alternative is to collect information from the user about his preferences and to use this information to build the user profile dynamically. This can be accomplished as follows.

In the very beginning, the user provides a set of keywords which describe an initial (and primitive) profile of his preferences. As new documents arrive, the system uses this profile to select documents which are potentially of interest and

shows them to the user. The user then goes through a relevance feedback cycle (see Chapter 5) in which he indicates not only the documents which are really relevant but also the documents which are non-relevant. The system uses this information to adjust the user profile description such that it reflects the new preferences just declared. Of course, with this procedure the profile is continually changing. Hopefully, however, it stabilizes after a while and no longer changes drastically (unless, of course, the user's interests shift suddenly). Chapter 5 illustrates mechanisms which can be used to dynamically update a keyword-based profile.

From the above, it should be clear that the filtering task can be viewed as a conventional information retrieval task in which the documents are the ones which keep arriving at the system. Ranking can be computed as before. The difficulty with filtering resides in describing appropriately the user's preferences in a user profile. The most common approaches for deriving a user profile are based on collecting relevant information from the user, deriving preferences from this information, and modifying the user profile accordingly. Since the number of potential applications of filtering keeps increasing, we should see in the future a renewed interest in the study and usage of the technique.

2.4 A Formal Characterization of IR Models

We have argued that the fundamental premises which form the basis for a ranking algorithm determine the IR model. Throughout this chapter, we will discuss different sets of such premises. However, before doing so, we should state clearly what exactly an IR model is. Our characterization is as follows.

Definition *An information retrieval* model *is a quadruple* $[\mathbf{D}, \mathbf{Q}, \mathcal{F}, R(q_i, d_j)]$ *where*

(1) $\mathbf{D}$ *is a set composed of logical views (or representations) for the documents in the collection.*

(2) $\mathbf{Q}$ *is a set composed of logical views (or representations) for the user information needs. Such representations are called* queries.

(3) $\mathcal{F}$ *is a framework for modeling document representations, queries, and their relationships.*

(4) $R(q_i, d_j)$ *is a ranking function which associates a real number with a query* $q_i \in \mathbf{Q}$ *and a document representation* $d_j \in \mathbf{D}$. *Such ranking defines an ordering among the documents with regard to the query* q_i.

To build a model, we think first of representations for the documents and for the user information need. Given these representations, we then conceive the framework in which they can be modeled. This framework should also provide the intuition for constructing a ranking function. For instance, for the classic Boolean model, the framework is composed of sets of documents and the standard operations on sets. For the classic vector model, the framework is composed of a

t-dimensional vectorial space and standard linear algebra operations on vectors. For the classic probabilistic model, the framework is composed of sets, standard probability operations, and the Bayes' theorem.

In the remainder of this chapter, we discuss the various IR models shown in Figure 2.1. Throughout the discussion, we do not explicitly instantiate the components $\mathbf{D}$, $\mathbf{Q}$, $\mathcal{F}$, and $R(q_i, d_j)$ of each model. Such components should be quite clear from the discussion and can be easily inferred.

2.5 Classic Information Retrieval

In this section we briefly present the three classic models in information retrieval namely, the Boolean, the vector, and the probabilistic models.

2.5.1 Basic Concepts

The classic models in information retrieval consider that each document is described by a set of representative keywords called index terms. An *index term* is simply a (document) word whose semantics helps in remembering the document's main themes. Thus, index terms are used to index and summarize the document contents. In general, index terms are mainly nouns because nouns have meaning by themselves and thus, their semantics is easier to identify and to grasp. Adjectives, adverbs, and connectives are less useful as index terms because they work mainly as complements. However, it might be interesting to consider all the distinct words in a document collection as index terms. For instance, this approach is adopted by some Web search engines as discussed in Chapter 13 (in which case, the document logical view is *full text*). We postpone a discussion on the problem of how to generate index terms until Chapter 7, where the issue is covered in detail.

Given a set of index terms for a document, we notice that not all terms are equally useful for describing the document contents. In fact, there are index terms which are simply vaguer than others. Deciding on the importance of a term for summarizing the contents of a document is not a trivial issue. Despite this difficulty, there are properties of an index term which are easily measured and which are useful for evaluating the potential of a term as such. For instance, consider a collection with a hundred thousand documents. A word which appears in each of the one hundred thousand documents is completely useless as an index term because it does not tell us anything about which documents the user might be interested in. On the other hand, a word which appears in just five documents is quite useful because it narrows down considerably the space of documents which might be of interest to the user. Thus, it should be clear that distinct index terms have varying relevance when used to describe document contents. This effect is captured through the assignment of numerical *weights* to each index term of a document.

Let k_i be an index term, d_j be a document, and $w_{i,j} \geq 0$ be a *weight* associated with the pair (k_i, d_j). This weight quantifies the importance of the index term for describing the document semantic contents.

Definition *Let t be the number of index terms in the system and k_i be a generic index term. $K = \{k_1, \ldots, k_t\}$ is the set of all index terms. A weight $w_{i,j} > 0$ is associated with each index term k_i of a document d_j. For an index term which does* not *appear in the document text, $w_{i,j} = 0$. With the document d_j is associated an index term vector $\vec{d_j}$ represented by $\vec{d_j} = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$. Further, let g_i be a function that returns the weight associated with the index term k_i in any t-dimensional vector (i.e., $g_i(\vec{d_j}) = w_{i,j}$).*

As we later discuss, the index term weights are usually assumed to be mutually independent. This means that knowing the weight $w_{i,j}$ associated with the pair (k_i, d_j) tells us nothing about the weight $w_{i+1,j}$ associated with the pair (k_{i+1}, d_j). This is clearly a simplification because occurrences of index terms in a document are not uncorrelated. Consider, for instance, that the terms *computer* and *network* are used to index a given document which covers the area of computer networks. Frequently, in this document, the appearance of one of these two words attracts the appearance of the other. Thus, these two words are correlated and their weights could reflect this correlation. While mutual independence seems to be a strong simplification, it does simplify the task of computing index term weights and allows for fast ranking computation. Furthermore, taking advantage of index term correlations for improving the final document ranking is not a simple task. In fact, none of the many approaches proposed in the past has clearly demonstrated that index term correlations are advantageous (for ranking purposes) with general collections. Therefore, unless clearly stated otherwise, we assume mutual independence among index terms. In Chapter 5 we discuss modern retrieval techniques which are based on term correlations and which have been tested successfully with particular collections. These successes seem to be slowly shifting the current understanding towards a more favorable view of the usefulness of term correlations for information retrieval systems.

The above definitions provide support for discussing the three classic information retrieval models, namely, the Boolean, the vector, and the probabilistic models, as we now do.

2.5.2 Boolean Model

The Boolean model is a simple retrieval model based on set theory and Boolean algebra. Since the concept of a set is quite intuitive, the Boolean model provides a framework which is easy to grasp by a common user of an IR system. Furthermore, the queries are specified as Boolean expressions which have precise semantics. Given its inherent simplicity and neat formalism, the Boolean model received great attention in past years and was adopted by many of the early commercial bibliographic systems.

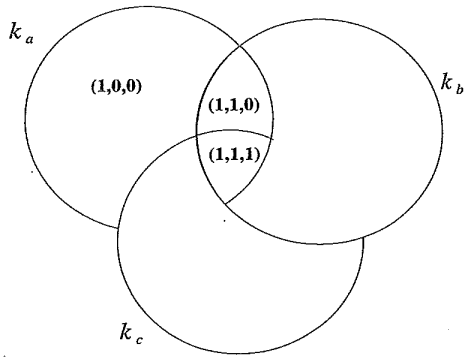


Figure 2.3 The three conjunctive components for the query $[q = k_a \wedge (k_b \vee \neg k_c)]$.

Unfortunately, the Boolean model suffers from major drawbacks. First, its retrieval strategy is based on a binary decision criterion (i.e., a document is predicted to be either relevant or non-relevant) without any notion of a grading scale, which prevents good retrieval performance. Thus, the Boolean model is in reality much more a data (instead of information) retrieval model. Second, while Boolean expressions have precise semantics, frequently it is not simple to translate an information need into a Boolean expression. In fact, most users find it difficult and awkward to express their query requests in terms of Boolean expressions. The Boolean expressions actually formulated by users often are quite simple (see Chapter 10 for a more thorough discussion on this issue). Despite these drawbacks, the Boolean model is still the dominant model with commercial document database systems and provides a good starting point for those new to the field.

The Boolean model considers that index terms are present or absent in a document. As a result, the index term weights are assumed to be all binary, i.e., $w_{i,j} \in \{0, 1\}$. A query q is composed of index terms linked by three connectives: *not*, *and*, *or*.Thus, a query is essentially a conventional Boolean expression which can be represented as a disjunction of conjunctive vectors (i.e., in *disjunctive normal form* — DNF). For instance, the query $[q = k_a \wedge (k_b \vee \neg k_c)]$ can be written in disjunctive normal form as $[\vec{q}_{dnf} = (1, 1, 1) \vee (1, 1, 0) \vee (1, 0, 0)]$, where each of the components is a binary weighted vector associated with the tuple (k_a, k_b, k_c). These binary weighted vectors are called the conjunctive components of $\vec{q}_{dnf}$. Figure 2.3 illustrates the three conjunctive components for the query q.

Definition *For the Boolean model, the index term weight variables are all binary i.e., $w_{i,j} \in \{0, 1\}$. A query q is a conventional Boolean expression. Let $\vec{q}_{dnf}$ be the disjunctive normal form for the query q. Further, let $\vec{q}_{cc}$ be any of the conjunctive components of $\vec{q}_{dnf}$. The similarity of a document d_j to the query q is defined as*

$$sim(d_j, q) = \begin{cases} 1 & \text{if } \exists \vec{q}_{cc} \mid (\vec{q}_{cc} \in \vec{q}_{dnf}) \wedge (\forall_{k_i},\ g_i(\vec{d_j}) = g_i(\vec{q}_{cc})) \\ 0 & \text{otherwise} \end{cases}$$

If $sim(d_j, q) = 1$ then the Boolean model predicts that the document d_j is relevant to the query q (it might not be). Otherwise, the prediction is that the document is not *relevant.*

The Boolean model predicts that each document is either *relevant* or *non-relevant.* There is no notion of a *partial match* to the query conditions. For instance, let d_j be a document for which $\vec{d_j} = (0, 1, 0)$. Document d_j includes the index term k_b but is considered non-relevant to the query $[q = k_a \wedge (k_b \vee \neg k_c)]$.

The main *advantages* of the Boolean model are the clean formalism behind the model and its simplicity. The main *disadvantages* are that exact matching may lead to retrieval of too few or too many documents (see Chapter 10). Today, it is well known that index term weighting can lead to a substantial improvement in retrieval performance. Index term weighting brings us to the vector model.

2.5.3 Vector Model

The vector model [697, 695] recognizes that the use of binary weights is too limiting and proposes a framework in which partial matching is possible. This is accomplished by assigning *non-binary* weights to index terms in queries and in documents. These term weights are ultimately used to compute the *degree of similarity* between each document stored in the system and the user query. By sorting the retrieved documents in decreasing order of this degree of similarity, the vector model takes into consideration documents which match the query terms only partially. The main resultant effect is that the ranked document answer set is a lot more precise (in the sense that it better matches the user information need) than the document answer set retrieved by the Boolean model.

Definition *For the vector model, the weight $w_{i,j}$ associated with a pair (k_i, d_j) is positive and non-binary. Further, the index terms in the query are also weighted. Let $w_{i,q}$ be the weight associated with the pair $[k_i, q]$, where $w_{i,q} \geq 0$. Then, the query vector $\vec{q}$ is defined as $\vec{q} = (w_{1,q}, w_{2,q}, \ldots, w_{t,q})$ where t is the total number of index terms in the system. As before, the vector for a document d_j is represented by $\vec{d_j} = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$.*

Therefore, a document d_j and a user query q are represented as t-dimensional vectors as shown in Figure 2.4. The vector model proposes to evaluate the degree of similarity of the document d_j with regard to the query q as the correlation between the vectors $\vec{d_j}$ and $\vec{q}$. This correlation can be quantified, for instance, by the *cosine of the angle* between these two vectors. That is,

$$
\begin{aligned}
sim(d_j, q) &= \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} \\
&= \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}
\end{aligned}
$$

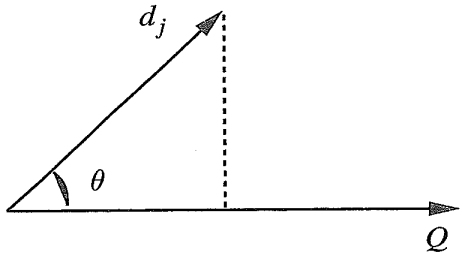


Figure 2.4 The cosine of θ is adopted as $sim(d_j, q)$.

where $|\vec{d_j}|$ and $|\vec{q}|$ are the norms of the document and query vectors. The factor $|\vec{q}|$ does not affect the ranking (i.e., the ordering of the documents) because it is the same for all documents. The factor $|\vec{d_j}|$ provides a normalization in the space of the documents.

Since $w_{i,j} \geq 0$ and $w_{i,q} \geq 0$, $sim(q, d_j)$ varies from 0 to +1. Thus, instead of attempting to predict whether a document is relevant or not, the vector model ranks the documents according to their *degree of similarity* to the query. A document might be retrieved even if it matches the query only *partially*. For instance, one can establish a threshold on $sim(d_j, q)$ and retrieve the documents with a degree of similarity above that threshold. But to compute rankings we need first to specify how index term weights are obtained.

Index term weights can be calculated in many different ways. The work by Salton and McGill [698] reviews various term-weighting techniques. Here, we do not discuss them in detail. Instead, we concentrate on elucidating the main idea behind the most effective term-weighting techniques. This idea is related to the basic principles which support clustering techniques, as follows.

Given a collection C of objects and a *vague* description of a set A, the goal of a simple clustering algorithm might be to separate the collection C of objects into two sets: a first one which is composed of objects related to the set A and a second one which is composed of objects not related to the set A. Vague description here means that we do not have complete information for deciding precisely which objects are and which are not in the set A. For instance, one might be looking for a set A of cars which have a price *comparable* to that of a Lexus 400. Since it is not clear what the term *comparable* means exactly, there is not a precise (and unique) description of the set A. More sophisticated clustering algorithms might attempt to separate the objects of a collection into various clusters (or classes) according to their properties. For instance, patients of a doctor specializing in cancer could be classified into five classes: terminal, advanced, metastasis, diagnosed, and healthy. Again, the possible class descriptions might be imprecise (and not unique) and the problem is one of deciding to which of these classes a new patient should be assigned. In what follows, however, we only discuss the simpler version of the clustering problem (i.e., the one which considers only two classes) because all that is required is a decision on which documents are predicted to be relevant and which ones are predicted to be not relevant (with regard to a given user query).

To view the IR problem as one of clustering, we refer to the early work of Salton. We think of the documents as a collection C of objects and think of the user query as a (vague) specification of a set A of objects. In this scenario, the IR problem can be reduced to the problem of determining which documents are in the set A and which ones are not (i.e., the IR problem can be viewed as a clustering problem). In a clustering problem, two main issues have to be resolved. First, one needs to determine what are the features which better describe the objects in the set A. Second, one needs to determine what are the features which better distinguish the objects in the set A from the remaining objects in the collection C. The first set of features provides for quantification of *intra-cluster* similarity, while the second set of features provides for quantification of *inter-cluster* dissimilarity. The most successful clustering algorithms try to balance these two effects.

In the vector model, intra-clustering similarity is quantified by measuring the raw frequency of a term k_i inside a document d_j. Such term frequency is usually referred to as the *tf factor* and provides one measure of how well that term describes the document contents (i.e., intra-document characterization). Furthermore, inter-cluster dissimilarity is quantified by measuring the inverse of the frequency of a term k_i among the documents in the collection. This factor is usually referred to as the *inverse document frequency* or the *idf factor*. The motivation for usage of an idf factor is that terms which appear in many documents are not very useful for distinguishing a relevant document from a non-relevant one. As with good clustering algorithms, the most effective term-weighting schemes for IR try to balance these two effects.

Definition *Let N be the total number of documents in the system and n_i be the number of documents in which the index term k_i appears. Let $freq_{i,j}$ be the raw frequency of term k_i in the document d_j (i.e., the number of times the term k_i is mentioned in the text of the document d_j). Then, the normalized frequency $f_{i,j}$ of term k_i in document d_j is given by*

$$f_{i,j} = \frac{freq_{i,j}}{max_l \ freq_{l,j}} \tag{2.1}$$

where the maximum is computed over all terms which are mentioned in the text of the document d_j. If the term k_i does not appear in the document d_j then $f_{i,j} = 0$. Further, let idf_i, inverse document frequency for k_i, be given by

$$idf_i = \log \frac{N}{n_i} \tag{2.2}$$

The best known term-weighting schemes use weights which are given by

$$w_{i,j} = f_{i,j} \times \log \frac{N}{n_i} \tag{2.3}$$

or by a variation of this formula. Such term-weighting strategies are called tf-idf schemes.

Several variations of the above expression for the weight $w_{i,j}$ are described in an interesting paper by Salton and Buckley which appeared in 1988 [696]. However, in general, the above expression should provide a good weighting scheme for many collections.

For the query term weights, Salton and Buckley suggest

$$w_{i,q} = \left(0.5 + \frac{0.5\ freq_{i,q}}{max_l\ freq_{l,q}}\right) \times \log \frac{N}{n_i} \tag{2.4}$$

where $freq_{i,q}$ is the raw frequency of the term k_i in the text of the information request q.

The main *advantages* of the vector model are: (1) its term-weighting scheme improves retrieval performance; (2) its partial matching strategy allows retrieval of documents that *approximate* the query conditions; and (3) its cosine ranking formula sorts the documents according to their degree of similarity to the query. Theoretically, the vector model has the *disadvantage* that index terms are assumed to be mutually independent (equation 2.3 does *not* account for index term dependencies). However, in practice, consideration of term dependencies might be a disadvantage. Due to the locality of many term dependencies, their indiscriminate application to all the documents in the collection might in fact *hurt* the overall performance.

Despite its simplicity, the vector model is a resilient ranking strategy with general collections. It yields ranked answer sets which are difficult to improve upon without query expansion or relevance feedback (see Chapter 5) within the framework of the vector model. A large variety of alternative ranking methods have been compared to the vector model but the consensus seems to be that, in general, the vector model is either superior or almost as good as the known alternatives. Furthermore, it is simple and fast. For these reasons, the vector model is a popular retrieval model nowadays.

2.5.4 Probabilistic Model

In this section, we describe the classic probabilistic model introduced in 1976 by Roberston and Sparck Jones [677] which later became known as the *binary independence retrieval* (BIR) model. Our discussion is intentionally brief and focuses mainly on highlighting the key features of the model. With this purpose in mind, we do not detain ourselves in subtleties regarding the binary independence assumption for the model. The section on bibliographic discussion points to references which cover these details.

The probabilistic model attempts to capture the IR problem within a probabilistic framework. The fundamental idea is as follows. Given a user query, there is a set of documents which contains exactly the relevant documents and

no other. Let us refer to this set of documents as the *ideal* answer set. Given the description of this ideal answer set, we would have no problems in retrieving its documents. Thus, we can think of the querying process as a process of specifying the properties of an ideal answer set (which is analogous to interpreting the IR problem as a problem of clustering). The problem is that we do not know exactly what these properties are. All we know is that there are index terms whose semantics should be used to characterize these properties. Since these properties are not known at query time, an effort has to be made at initially guessing what they could be. This initial guess allows us to generate a preliminary probabilistic description of the ideal answer set which is used to retrieve a first set of documents. An interaction with the user is then initiated with the purpose of improving the probabilistic description of the ideal answer set. Such interaction could proceed as follows.

The user takes a look at the retrieved documents and decides which ones are relevant and which ones are not (in truth, only the first top documents need to be examined). The system then uses this information to refine the description of the ideal answer set. By repeating this process many times, it is expected that such a description will evolve and become closer to the real description of the ideal answer set. Thus, one should always have in mind the need to guess at the beginning the description of the ideal answer set. Furthermore, a conscious effort is made to model this description in probabilistic terms.

The probabilistic model is based on the following fundamental assumption.

Assumption (Probabilistic Principle) Given a user query q and a document d_j in the collection, the probabilistic model tries to estimate the probability that the user will find the document d_j interesting (i.e., relevant). The model assumes that this probability of relevance depends on the query and the document representations only. Further, the model assumes that there is a subset of all documents which the user prefers as the answer set for the query q. Such an *ideal* answer set is labeled R and should maximize the overall probability of relevance to the user. Documents in the set R are predicted to be *relevant* to the query. Documents not in this set are predicted to be *non-relevant.*

This assumption is quite troublesome because it does not state explicitly how to compute the probabilities of relevance. In fact, not even the sample space which is to be used for defining such probabilities is given.

Given a query q, the probabilistic model assigns to each document d_j, as a measure of its similarity to the query, the ratio $P(d_j \text{ relevant-to } q)/P(d_j \text{ non-relevant-to } q)$ which computes the odds of the document d_j being relevant to the query q. Taking the odds of relevance as the rank minimizes the probability of an erroneous judgement [282, 785].

Definition *For the probabilistic model, the index term weight variables are all binary i.e.,* $w_{i,j} \in \{0,1\}$, $w_{i,q} \in \{0,1\}$. *A query* q *is a subset of index terms. Let* R *be the set of documents known (or initially guessed) to be relevant. Let* $\overline{R}$ *be the complement of* R *(i.e., the set of non-relevant documents). Let* $P(R|\vec{d}_j)$

be the probability that the document d_j is relevant to the query q and $P(\overline{R}|\vec{d_j})$ be the probability that d_j is non-relevant to q. The similarity $sim(d_j, q)$ of the document d_j to the query q is defined as the ratio

$$sim(d_j, q) = \frac{P(R|\vec{d_j})}{P(\overline{R}|\vec{d_j})}$$

Using Bayes' rule,

$$sim(d_j, q) = \frac{P(\vec{d_j}|R) \times P(R)}{P(\vec{d_j}|\overline{R}) \times P(\overline{R})}$$

$P(\vec{d_j}|R)$ stands for the probability of randomly selecting the document d_j from the set R of relevant documents. Further, $P(R)$ stands for the probability that a document randomly selected from the entire collection is relevant. The meanings attached to $P(\vec{d_j}|\overline{R})$ and $P(\overline{R})$ are analogous and complementary.

Since $P(R)$ and $P(\overline{R})$ are the same for all the documents in the collection, we write,

$$sim(d_j, q) \sim \frac{P(\vec{d_j}|R)}{P(\vec{d_j}|\overline{R})}$$

Assuming independence of index terms,

$$sim(d_j, q) \sim \frac{(\prod_{g_i(\vec{d_j})=1} P(k_i|R)) \times (\prod_{g_i(\vec{d_j})=0} P(\overline{k}_i|R))}{(\prod_{g_i(\vec{d_j})=1} P(k_i|\overline{R})) \times (\prod_{g_i(\vec{d_j})=0} P(\overline{k}_i|\overline{R}))}$$

$P(k_i|R)$ stands for the probability that the index term k_i is present in a document randomly selected from the set R. $P(\overline{k}_i|R)$ stands for the probability that the index term k_i is not present in a document randomly selected from the set R. The probabilities associated with the set $\overline{R}$ have meanings which are analogous to the ones just described.

Taking logarithms, recalling that $P(k_i|R) + P(\overline{k}_i|R) = 1$, and ignoring factors which are constant for all documents in the context of the same query, we can finally write

$$sim(d_j, q) \sim \sum_{i=1}^{t} w_{i,q} \times w_{i,j} \times \left(\log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\overline{R})}{P(k_i|\overline{R})} \right)$$

which is a key expression for ranking computation in the probabilistic model.

Since we do not know the set R at the beginning, it is necessary to devise a method for initially computing the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$. There are many alternatives for such computation. We discuss a couple of them below.

In the very beginning (i.e., immediately after the query specification), there are no retrieved documents. Thus, one has to make simplifying assumptions such as: (a) assume that $P(k_i|R)$ is constant for all index terms k_i (typically, equal to 0.5) and (b) assume that the distribution of index terms among the non-relevant documents can be approximated by the distribution of index terms among all the documents in the collection. These two assumptions yield

$$\begin{aligned} P(k_i|R) &= 0.5 \\ P(k_i|\overline{R}) &= \frac{n_i}{N} \end{aligned}$$

where, as already defined, n_i is the number of documents which contain the index term k_i and N is the total number of documents in the collection. Given this initial guess, we can then retrieve documents which contain query terms and provide an initial probabilistic ranking for them. After that, this initial ranking is improved as follows.

Let V be a subset of the documents initially retrieved and ranked by the probabilistic model. Such a subset can be defined, for instance, as the top r ranked documents where r is a previously defined threshold. Further, let V_i be the subset of V composed of the documents in V which contain the index term k_i. For simplicity, we also use V and V_i to refer to the number of elements in these sets (it should always be clear when the used variable refers to the set or to the number of elements in it). For improving the probabilistic ranking, we need to improve our guesses for $P(k_i|R)$ and $P(k_i|\overline{R})$. This can be accomplished with the following assumptions: (a) we can approximate $P(k_i|R)$ by the distribution of the index term k_i among the documents retrieved so far, and (b) we can approximate $P(k_i|\overline{R})$ by considering that all the non-retrieved documents are not relevant. With these assumptions, we can write,

$$\begin{aligned} P(k_i|R) &= \frac{V_i}{V} \\ P(k_i|\overline{R}) &= \frac{n_i - V_i}{N - V} \end{aligned}$$

This process can then be repeated recursively. By doing so, we are able to improve on our guesses for the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$ without any assistance from a human subject (contrary to the original idea). However, we can also use assistance from the user for definition of the subset V as originally conceived.

The last formulas for $P(k_i|R)$ and $P(k_i|\overline{R})$ pose problems for small values of V and V_i which arise in practice (such as $V = 1$ and $V_i = 0$). To circumvent these problems, an adjustment factor is often added in which yields

$$\begin{aligned} P(k_i|R) &= \frac{V_i + 0.5}{V + 1} \\ P(k_i|\overline{R}) &= \frac{n_i - V_i + 0.5}{N - V + 1} \end{aligned}$$

An adjustment factor which is constant and equal to 0.5 is not always satisfactory. An alternative is to take the fraction n_i/N as the adjustment factor which yields

$$P(k_i|R) = \frac{V_i + \frac{n_i}{N}}{V+1}$$
$$P(k_i|\overline{R}) = \frac{n_i - V_i + \frac{n_i}{N}}{N-V+1}$$

This completes our discussion of the probabilistic model.

The main *advantage* of the probabilistic model, in theory, is that documents are ranked in decreasing order of their *probability* of being relevant. The *disadvantages* include: (1) the need to guess the initial separation of documents into relevant and non-relevant sets; (2) the fact that the method does *not* take into account the frequency with which an index term occurs inside a document (i.e., all weights are binary); and (3) the adoption of the independence assumption for index terms. However, as discussed for the vector model, it is not clear that independence of index terms is a bad assumption in practical situations.

2.5.5 Brief Comparison of Classic Models

In general, the Boolean model is considered to be the weakest classic method. Its main problem is the inability to recognize partial matches which frequently leads to poor performance. There is some controversy as to whether the probabilistic model outperforms the vector model. Croft performed some experiments and suggested that the probabilistic model provides a better retrieval performance. However, experiments done afterwards by Salton and Buckley refute that claim. Through several different measures, Salton and Buckley showed that the vector model is *expected* to outperform the probabilistic model with general collections. This also seems to be the dominant thought among researchers, practitioners, and the Web community, where the popularity of the vector model runs high.

2.6 Alternative Set Theoretic Models

In this section, we discuss two alternative set theoretic models, namely the fuzzy set model and the extended Boolean model.

2.6.1 Fuzzy Set Model

Representing documents and queries through sets of keywords yields descriptions which are only partially related to the real semantic contents of the respective documents and queries. As a result, the matching of a document to the query terms is approximate (or vague). This can be modeled by considering that each

query term defines a *fuzzy* set and that each document has a *degree of membership* (usually smaller than 1) in this set. This interpretation of the retrieval process (in terms of concepts from fuzzy theory) is the basic foundation of the various fuzzy set models for information retrieval which have been proposed over the years. Instead of reviewing several of these models here, we focus on a particular one whose description fits well with the models already covered in this chapter. Thus, our discussion is based on the fuzzy set model for information retrieval proposed by Ogawa, Morita, and Kobayashi [616]. Before proceeding, we briefly introduce some fundamental concepts.

Fuzzy Set Theory

Fuzzy set theory [846] deals with the representation of classes whose boundaries are not well defined. The key idea is to associate a membership function with the elements of the class. This function takes values in the interval [0,1] with 0 corresponding to no membership in the class and 1 corresponding to full membership. Membership values between 0 and 1 indicate *marginal* elements of the class. Thus, membership in a fuzzy set is a notion intrinsically *gradual* instead of abrupt (as in conventional Boolean logic).

Definition *A fuzzy subset A of a universe of discourse U is characterized by a membership function $\mu_A : U \rightarrow [0, 1]$ which associates with each element u of U a number $\mu_A(u)$ in the interval [0,1].*

The three most commonly used operations on fuzzy sets are: the *complement* of a fuzzy set, the *union* of two or more fuzzy sets, and the *intersection* of two or more fuzzy sets. They are defined as follows.

Definition *Let U be the universe of discourse, A and B be two fuzzy subsets of U, and $\overline{A}$ be the complement of A relative to U. Also, let u be an element of U. Then,*

$$\begin{aligned} \mu_{\overline{A}}(u) &= 1 - \mu_A(u) \\ \mu_{A \cup B}(u) &= max(\mu_A(u), \mu_B(u)) \\ \mu_{A \cap B}(u) &= min(\mu_A(u), \mu_B(u)) \end{aligned}$$

Fuzzy sets are useful for representing vagueness and imprecision and have been applied to various domains. In what follows, we discuss their application to information retrieval.

Fuzzy Information Retrieval

As discussed in Chapters 5 and 7, one additional approach to modeling the information retrieval process is to adopt a thesaurus (which defines term re-

lationships). The basic idea is to expand the set of index terms in the query with related terms (obtained from the thesaurus) such that additional relevant documents (i.e., besides the ones which would be normally retrieved) can be retrieved by the user query. A thesaurus can also be used to model the information retrieval problem in terms of fuzzy sets as follows.

A thesaurus can be constructed by defining a *term-term correlation matrix* $\vec{c}$ (called *keyword connection matrix* in [616]) whose rows and columns are associated to the index terms in the document collection. In this matrix $\vec{c}$, a normalized correlation factor $c_{i,l}$ between two terms k_i and k_l can be defined by

$$c_{i,l} = \frac{n_{i,l}}{n_i + n_l - n_{i,l}}$$

where n_i is the number of documents which contain the term k_i, n_l is the number of documents which contain the term k_l, and $n_{i,l}$ is the number of documents which contain both terms. Such a correlation metric is quite common and has been used extensively with clustering algorithms as detailed in Chapter 5.

We can use the term correlation matrix $\vec{c}$ to define a fuzzy set associated to each index term k_i. In this fuzzy set, a document d_j has a degree of membership $\mu_{i,j}$ computed as

$$\mu_{i,j} = 1 - \prod_{k_l \in d_j} (1 - c_{i,l})$$

which computes an algebraic sum (here implemented as the complement of a negated algebraic product) over all terms in the document d_j. A document d_j belongs to the fuzzy set associated to the term k_i if its own terms are related to k_i. Whenever there is at least one index term k_l of d_j which is strongly related to the index k_i (i.e., $c_{i,l} \sim 1$), then $\mu_{i,j} \sim 1$ and the index k_i is a good fuzzy index for the document d_j. In the case when all index terms of d_j are only loosely related to k_i, the index k_i is not a good fuzzy index for d_j (i.e., $\mu_{i,j} \sim 0$). The adoption of an algebraic sum over all terms in the document d_j (instead of the classic max function) allows a smooth transition for the values of the $\mu_{i,j}$ factor.

The user states his information need by providing a Boolean-like query expression. As also happens with the classic Boolean model (see the beginning of this chapter), this query is converted to its disjunctive normal form. For instance, the query $[q = k_a \wedge (k_b \vee \neg k_c)]$ can be written in disjunctive normal form as $[\vec{q}_{dnf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)]$, where each of the components is a binary weighted vector associated to the tuple (k_a, k_b, k_c). These binary weighted vectors are the conjunctive components of $\vec{q}_{dnf}$. Let cc_i be a reference to the i-th conjunctive component. Then,

$$\vec{q}_{dnf} = cc_1 \vee cc_2 \vee \ldots \vee cc_p$$

where p is the number of conjunctive components of $\vec{q}_{dnf}$. The procedure to compute the documents relevant to a query is analogous to the procedure adopted

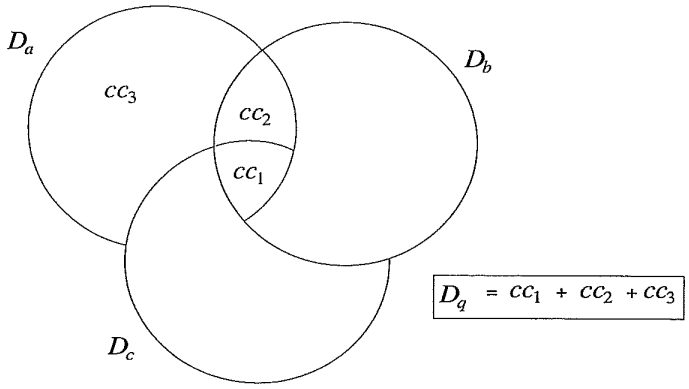


Figure 2.5 Fuzzy document sets for the query $[q = k_a \wedge (k_b \vee \neg k_c)]$. Each cc_i, $i \in \{1, 2, 3\}$, is a conjunctive component. D_q is the query fuzzy set.

by the classic Boolean model. The difference is that here we deal with fuzzy (instead of *crispy* or Boolean) sets. We proceed with an example.

Consider again the query $[q = k_a \wedge (k_b \vee \neg k_c)]$. Let D_a be the fuzzy set of documents associated to the index k_a. This set is composed, for instance, by the documents d_j which have a degree of membership $\mu_{a,j}$ greater than a predefined threshold K. Further, let $\overline{D}_a$ be the complement of the set D_a. The fuzzy set $\overline{D}_a$ is associated to $\overline{k}_a$, the negation of the index term k_a. Analogously, we can define fuzzy sets D_b and D_c associated to the index terms k_b and k_c, respectively. Figure 2.5 illustrates this example. Since the sets are all fuzzy, a document d_j might belong to the set D_a, for instance, even if the text of the document d_j does not mention the index k_a.

The query fuzzy set D_q is a union of the fuzzy sets associated with the three conjunctive components of $\vec{q}_{dnf}$ (which are referred to as cc_1, cc_2, and cc_3). The membership $\mu_{q,j}$ of a document d_j in the fuzzy answer set D_q is computed as follows.

$$\begin{aligned}
\mu_{q,j} &= \mu_{cc_1+cc_2+cc_3,j} \\
&= 1 - \prod_{i=1}^{3}(1 - \mu_{cc_i,j}) \\
&= 1 - (1 - \mu_{a,j}\mu_{b,j}\mu_{c,j}) \times \\
&\quad (1 - \mu_{a,j}\mu_{b,j}(1 - \mu_{c,j})) \times (1 - \mu_{a,j}(1 - \mu_{b,j})(1 - \mu_{c,j}))
\end{aligned}$$

where $\mu_{i,j}$, $i \in \{a, b, c\}$, is the membership of d_j in the fuzzy set associated with k_i.

As already observed, the degree of membership in a disjunctive fuzzy set is computed here using an *algebraic sum*, instead of the more common *max* function. Further, the degree of membership in a conjunctive fuzzy set is computed here using an *algebraic product*, instead of the more common *min* function. This adoption of algebraic sums and products yields degrees of membership which

vary more smoothly than those computed using the *min* and *max* functions and thus seem more appropriate to an information retrieval system.

This example illustrates how this fuzzy model ranks documents relative to the user query. The model uses a term-term correlation matrix to compute correlations between a document d_j and its fuzzy index terms. Further, the model adopts algebraic sums and products (instead of *max* and *min*) to compute the overall degree of membership of a document d_j in the fuzzy set defined by the user query. Ogawa, Morita, and Kobayashi [616] also discuss how to incorporate user relevance feedback into the model but such discussion is beyond the scope of this chapter.

Fuzzy set models for information retrieval have been discussed mainly in the literature dedicated to fuzzy theory and are not popular among the information retrieval community. Further, the vast majority of the experiments with fuzzy set models has considered only small collections which make comparisons difficult to make at this time.

2.6.2 Extended Boolean Model

Boolean retrieval is simple and elegant. However, since there is no provision for term weighting, no ranking of the answer set is generated. As a result, the size of the output might be too large or too small (see Chapter 10 for details on this issue). Because of these problems, modern information retrieval systems are no longer based on the Boolean model. In fact, most of the new systems adopt at their core some form of vector retrieval. The reasons are that the vector space model is simple, fast, and yields better retrieval performance. One alternative approach though is to extend the Boolean model with the functionality of partial matching and term weighting. This strategy allows one to combine Boolean query formulations with characteristics of the vector model. In what follows, we discuss one of the various models which are based on the idea of extending the Boolean model with features of the vector model.

The *extended Boolean model*, introduced in 1983 by Salton, Fox, and Wu [703], is based on a critique of a basic assumption in Boolean logic as follows. Consider a conjunctive Boolean query given by $q = k_x \wedge k_y$. According to the Boolean model, a document which contains either the term k_x or the term k_y is as irrelevant as another document which contains neither of them. However, this binary decision criteria frequently is not in accordance with common sense. An analogous reasoning applies when one considers purely disjunctive queries.

When only two terms are considered, we can plot queries and documents in a two-dimensional map as shown in Figure 2.6. A document d_j is positioned in this space through the adoption of weights $w_{x,j}$ and $w_{y,j}$ associated with the pairs $[k_x, d_j]$ and $[k_y, d_j]$, respectively. We assume that these weights are normalized and thus lie between 0 and 1. For instance, these weights can be computed as normalized tf-idf factors as follows.

$$w_{x,j} = f_{x,j} \times \frac{idf_x}{max_i \ idf_i}$$

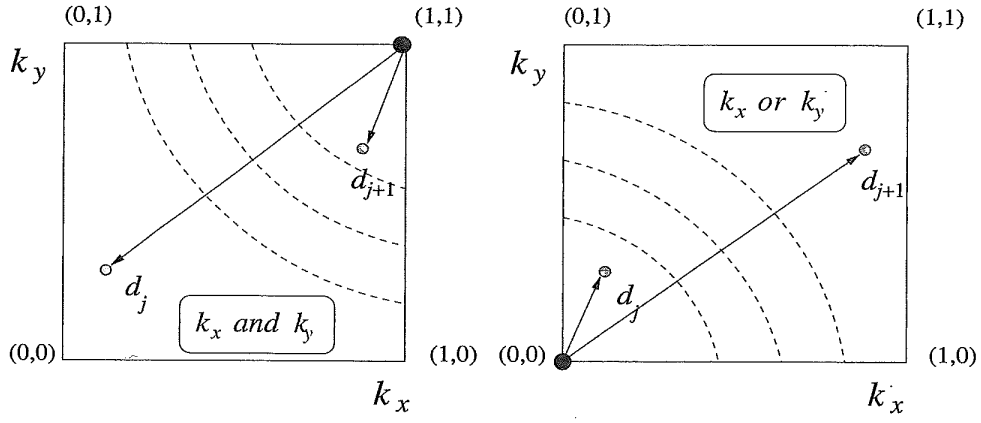


Figure 2.6 Extended Boolean logic considering the space composed of two terms k_x and k_y only.

where, as defined by equation 2.3, $f_{x,j}$ is the normalized frequency of term k_x in document d_j and idf_i is the inverse document frequency for a generic term k_i. For simplicity, in the remainder of this section, we refer to the weight $w_{x,j}$ as x, to the weight $w_{y,j}$ as y, and to the document vector $\vec{d_j} = (w_{x,j}, w_{y,j})$ as the point $d_j = (x, y)$. Observing Figure 2.6 we notice two particularities. First, for a disjunctive query $q_{or} = k_x \vee k_y$, the point $(0, 0)$ is the spot to be avoided. This suggests taking the distance from $(0, 0)$ as a measure of similarity with regard to the query q_{or}. Second, for a conjunctive query $q_{and} = k_x \wedge k_y$, the point $(1, 1)$ is the most desirable spot. This suggests taking the complement of the distance from the point $(1, 1)$ as a measure of similarity with regard to the query q_{and}. Furthermore, such distances can be normalized which yields,

$$sim(q_{or}, d) = \sqrt{\frac{x^2 + y^2}{2}}$$

$$sim(q_{and}, d) = 1 - \sqrt{\frac{(1-x)^2 + (1-y)^2}{2}}$$

If the weights are all Boolean (i.e., $w_{x,j} \in \{0, 1\}$), a document is always positioned in one of the four corners (i.e., (0,0), (0,1), (1,0), or (1,1)) and the values for $sim(q_{or}, d)$ are restricted to 0, $1/\sqrt{2}$, and 1. Analogously, the values for $sim(q_{and}, d)$ are restricted to 0, $1 - 1/\sqrt{2}$, and 1.

Given that the number of index terms in a document collection is t, the Boolean model discussed above can be naturally extended to consider Euclidean distances in a t-dimensional space. However, a more comprehensive generalization is to adopt the theory of vector norms as follows.

The *p-norm model* generalizes the notion of distance to include not only Euclidean distances but also p-distances, where $1 \leq p \leq \infty$ is a newly introduced parameter whose value must be specified at query time. A generalized disjunctive

query is now represented by

$$q_{or} = k_1 \ \vee^p \ k_2 \ \vee^p \ \ldots \vee^p k_m$$

Analogously, a generalized conjunctive query is now represented by

$$q_{and} = k_1 \ \wedge^p \ k_2 \ \wedge^p \ \ldots \wedge^p k_m$$

The respective query-document similarities are now given by

$$sim(q_{or}, d_j) = \left(\frac{x_1^p + x_2^p + \ldots + x_m^p}{m} \right)^{\frac{1}{p}}$$

$$sim(q_{and}, d_j) = 1 - \left(\frac{(1 - x_1)^p + (1 - x_2)^p + \ldots + (1 - x_m)^p}{m} \right)^{\frac{1}{p}}$$

where each x_i stands for the weight $w_{i,d}$ associated to the pair $[k_i, d_j]$.

The p norm as defined above enjoys a couple of interesting properties as follows. First, when $p = 1$ it can be verified that

$$sim(q_{or}, d_j) = sim(q_{and}, d_j) = \frac{x_1 + \ldots + x_m}{m}$$

Second, when $p = \infty$ it can be verified that

$$sim(q_{or}, d_j) = max(x_i)$$
$$sim(q_{and}, d_j) = min(x_i)$$

Thus, for $p = 1$, conjunctive and disjunctive queries are evaluated by a sum of term-document weights as done by vector-based similarity formulas (which compute the inner product). Further, for $p = \infty$, queries are evaluated according to the formalism of fuzzy logic (which we view as a generalization of Boolean logic). By varying the parameter p between 1 and infinity, we can vary the p-norm ranking behavior from that of a vector-like ranking to that of a Boolean-like ranking. This is quite powerful and is a good argument in favor of the extended Boolean model.

The processing of more general queries is done by grouping the operators in a predefined order. For instance, consider the query $q = (k_1 \wedge^p k_2) \vee^p k_3$. The similarity $sim(q, d_j)$ between a document d_j and this query is then computed as

$$sim(q, d) = \left(\frac{\left(1 - \left(\frac{(1-x_1)^p + (1-x_2)^p}{2}\right)^{\frac{1}{p}}\right)^p + x_3^p}{2} \right)^{\frac{1}{p}}$$

This procedure can be applied recursively no matter the number of AND/OR operators.

One additional interesting aspect of this extended Boolean model is the possibility of using combinations of different values of the parameter p in a same query request. For instance, the query

$$(k_1 \vee^2 k_2) \wedge^\infty k_3$$

could be used to indicate that k_1 and k_2 should be used as in a vector system but that the presence of k_3 is required (i.e., the conjunction is interpreted as a Boolean operation). Despite the fact that it is not clear whether this additional functionality has any practical impact, the model does allow for it and does so in a natural way (without the need for clumsy extensions to handle special cases).

We should also observe that the extended Boolean model relaxes Boolean algebra interpreting Boolean operations in terms of algebraic distances. In this sense, it is really a hybrid model which includes properties of both the set theoretic models and the algebraic models. For simplicity, we opted for classifying the model as a set theoretic one.

The extended Boolean model was introduced in 1983 but has not been used extensively. However, the model does provide a neat framework and might reveal itself useful in the future.

2.7 Alternative Algebraic Models

In this section, we discuss three alternative algebraic models namely, the generalized vector space model, the latent semantic indexing model, and the neural network model.

2.7.1 Generalized Vector Space Model

As already discussed, the three classic models assume independence of index terms. For the vector model, this assumption is normally interpreted as follows.

Definition *Let $\vec{k}_i$ be a vector associated with the index term k_i. Independence of index terms in the vector model implies that the set of vectors $\{\vec{k}_1, \vec{k}_2, \ldots, \vec{k}_t\}$ is linearly independent and forms a basis for the subspace of interest. The dimension of this space is the number t of index terms in the collection.*

Frequently, independence among index terms is interpreted in a more restrictive sense to mean pairwise orthogonality among the index term vectors i.e., to mean that for each pair of index term vectors $\vec{k}_i$ and $\vec{k}_j$ we have $\vec{k}_i \bullet \vec{k}_j = 0$. In 1985, however, Wong, Ziarko, and Wong [832] proposed an interpretation in which the index term vectors are assumed linearly independent but are not pairwise orthogonal. Such interpretation leads to the *generalized vector space model* which we now discuss.

In the generalized vector space model, two index term vectors might be non-orthogonal. This means that index term vectors are not seen as the orthogonal vectors which compose the basis of the space. Instead, they are themselves composed of *smaller* components which are derived from the particular collection at hand as follows.

Definition *Given the set* $\{k_1, k_2, \ldots, k_t\}$ *of index terms in a collection, as before, let* $w_{i,j}$ *be the weight associated with the term-document pair* $[k_i, d_j]$. *If the* $w_{i,j}$ *weights are all binary then all possible patterns of term co-occurrence (inside documents) can be represented by a set of* 2^t minterms *given by* $m_1 = (0, 0, \ldots, 0)$, $m_2 = (1, 0, \ldots, 0)$, $\ldots$, $m_{2^t} = (1, 1, \ldots, 1)$. *Let* $g_i(m_j)$ *return the weight* $\{0,1\}$ *of the index term* k_i *in the minterm* m_j.

Thus, the minterm m_1 (for which $g_i(m_1) = 0$, for all i) points to the documents containing none of the index terms. The minterm m_2 (for which $g_1(m_2) = 1$, for $i = 1$, and $g_i(m_2) = 0$, for $i > 1$) points to the documents containing solely the index term k_1. Further, the minterm m_{2^t} points to the documents containing all the index terms. The central idea in the generalized vector space model is to introduce a set of pairwise orthogonal vectors $\vec{m}_i$ associated with the set of minterms and to adopt this set of vectors as the basis for the subspace of interest.

Definition *Let us define the following set of* $\vec{m}_i$ *vectors*

$$\begin{aligned} \vec{m}_1 &= (1, 0, \ldots, 0, 0) \\ \vec{m}_2 &= (0, 1, \ldots, 0, 0) \\ &\vdots \\ \vec{m}_{2^t} &= (0, 0, \ldots, 0, 1) \end{aligned}$$

where each vector $\vec{m}_i$ *is associated with the respective minterm* m_i.

Notice that $\vec{m}_i \bullet \vec{m}_j = 0$ for all $i \neq j$ and thus the set of $\vec{m}_i$ vectors is, by definition, *pairwise* orthogonal. This set of $\vec{m}_i$ vectors is then taken as the orthonormal basis for the generalized vector space model.

Pairwise orthogonality among the $\vec{m}_i$ vectors does not imply independence among the index terms. On the contrary, index terms are now correlated by the $\vec{m}_i$ vectors. For instance, the vector $\vec{m}_4$ is associated with the minterm $m_4 = (1, 1, \ldots, 0)$ which points to the documents in the collection containing the index terms k_1, k_2, and no others. If such documents do exist in the collection under consideration then we say that the minterm m_4 is *active* and that a dependence between the index terms k_1 and k_2 is induced. If we consider this point more carefully, we notice that the generalized vector model adopts as a basic foundation the idea that co-occurrence of index terms inside documents in the collection induces dependencies among these index terms. Since this is an idea which was introduced many years before the generalized vector space

model itself, novelty is not granted. Instead, the main contribution of the model is the establishment of a formal framework in which dependencies among index terms (induced by co-occurrence patterns inside documents) can be nicely represented.

The usage of index term dependencies to improve retrieval performance continues to be a controversial issue. In fact, despite the introduction in the 1980s of more effective algorithms for incorporating term dependencies (see Chapter 5), there is no consensus that incorporation of term dependencies in the model yields effective improvement with general collections. Thus, it is not clear that the framework of the generalized vector model provides a clear advantage in practical situations. Further, the generalized vector model is more complex and computationally more expensive than the classic vector model.

To determine the index term vector $\vec{k}_i$ associated with the index term k_i, we simply sum up the vectors for all minterms m_r in which the term k_i is in state 1 and normalize. Thus,

$$\vec{k}_i = \frac{\sum_{\forall r,\ g_i(m_r)=1} c_{i,r}\ \vec{m}_r}{\sqrt{\sum_{\forall r,\ g_i(m_r)=1} c_{i,r}^2}} \tag{2.5}$$

$$c_{i,r} = \sum_{d_j \mid g_l(\vec{d_j})=g_l(m_r)\ for\ all\ l} w_{i,j}$$

These equations provide a general definition for the index term vector $\vec{k}_i$ in terms of the $\vec{m}_r$ vectors. The term vector $\vec{k}_i$ collects all the $\vec{m}_r$ vectors in which the index term k_i is in state 1. For each $\vec{m}_r$ vector, a correlation factor $c_{i,r}$ is defined. Such a correlation factor sums up the weights $w_{i,j}$ associated with the index term k_i and each document d_j whose term occurrence pattern coincides exactly with that of the minterm m_r. Thus, a minterm is of interest (in which case it is said to be *active*) only if there is at least one document in the collection which matches its term occurrence pattern. This implies that no more than N minterms can be active, where N is the number of documents in the collection. Therefore, the ranking computation does not depend on an exponential number of minterms as equation 2.5 seems to suggest.

Notice that the internal product $\vec{k}_i \bullet \vec{k}_j$ can now be used to quantify a degree of correlation between the index terms k_i and k_j. For instance,

$$\vec{k}_i \bullet \vec{k}_j = \sum_{\forall r \mid g_i(m_r)=1 \wedge g_j(m_r)=1} c_{i,r} \times c_{j,r}$$

which, as later discussed in Chapter 5, is a good technique for quantifying index term correlations.

In the classic vector model, a document d_j and a user query q are expressed by $\vec{d}_j = \sum_{\forall i} w_{i,j}\ \vec{k}_i$ and $\vec{q} = \sum_{\forall i} w_{i,q}\ \vec{k}_i$, respectively. In the generalized vector space model, these representations can be directly translated to the space of minterm vectors $\vec{m}_r$ by applying equation 2.5. The resultant $\vec{d}_j$ and $\vec{q}$ vectors

are then used for computing the ranking through a standard cosine similarity function.

The ranking that results from the generalized vector space model combines the standard $w_{i,j}$ term-document weights with the correlation factors $c_{i,r}$. However, since the usage of term-term correlations does not necessarily yield improved retrieval performance, it is not clear in which situations the generalized model outperforms the classic vector model. Furthermore, the cost of computing the ranking in the generalized model can be fairly high with large collections because, in this case, the number of active minterms (i.e., those which have to be considered for computing the $\vec{k_i}$ vectors) might be proportional to the number of documents in the collection. Despite these drawbacks, the generalized vector model does introduce new ideas which are of importance from a theoretical point of view.

2.7.2 Latent Semantic Indexing Model

As discussed earlier, summarizing the contents of documents and queries through a set of index terms can lead to poor retrieval performance due to two effects. First, many unrelated documents might be included in the answer set. Second, relevant documents which are not indexed by any of the query keywords are not retrieved. The main reason for these two effects is the inherent vagueness associated with a retrieval process which is based on keyword sets.

The ideas in a text are more related to the concepts described in it than to the index terms used in its description. Thus, the process of matching documents to a given query could be based on concept matching instead of index term matching. This would allow the retrieval of documents even when they are not indexed by query index terms. For instance, a document could be retrieved because it shares concepts with another document which is relevant to the given query. Latent semantic indexing is an approach introduced in 1988 which addresses these issues (for clustering-based approaches which also address these issues, see Chapter 5).

The main idea in the *latent semantic indexing model* [287] is to map each document and query vector into a lower dimensional space which is associated with concepts. This is accomplished by mapping the index term vectors into this lower dimensional space. The claim is that retrieval in the reduced space may be superior to retrieval in the space of index terms. Before proceeding, let us define basic terminology.

Definition *As before, let t be the number of index terms in the collection and N be the total number of documents. Define $\vec{M}=(M_{ij})$ as a term-document association matrix with t rows and N columns. To each element M_{ij} of this matrix is assigned a weight $w_{i,j}$ associated with the term-document pair $[k_i, d_j]$. This $w_{i,j}$ weight could be generated using the tf-idf weighting technique common in the classic vector space model.*

Latent semantic indexing proposes to decompose the $\vec{M}$ association matrix in three components using singular value decomposition as follows.

$$\vec{M} = \vec{K}\vec{S}\vec{D}^t$$

The matrix $\vec{K}$ is the matrix of eigenvectors derived from the term-to-term correlation matrix given by $\vec{M}\vec{M}^t$ (see Chapter 5). The matrix $\vec{D}^t$ is the matrix of eigenvectors derived from the transpose of the document-to-document matrix given by $\vec{M}^t\vec{M}$. The matrix $\vec{S}$ is an $r \times r$ diagonal matrix of singular values where $r = min(t, N)$ is the *rank* of $\vec{M}$.

Consider now that only the s largest singular values of $\vec{S}$ are kept along with their corresponding columns in $\vec{K}$ and $\vec{D}^t$ (i.e., the remaining singular values of $\vec{S}$ are deleted). The resultant $\vec{M}_s$ matrix is the matrix of rank s which is closest to the original matrix $\vec{M}$ in the least square sense. This matrix is given by

$$\vec{M}_s = \vec{K}_s\vec{S}_s\vec{D}_s^t$$

where s, $s < r$, is the dimensionality of a reduced concept space. The selection of a value for s attempts to balance two opposing effects. First, s should be large enough to allow fitting all the structure in the real data. Second, s should be small enough to allow filtering out all the non-relevant representational details (which are present in the conventional index-term based representation).

The relationship between any two documents in the reduced space of dimensionality s can be obtained from the $\vec{M}_s^t\vec{M}_s$ matrix given by

$$\begin{aligned}
\vec{M}_s^t\vec{M}_s &= (\vec{K}_s\vec{S}_s\vec{D}_s^t)^t\vec{K}_s\vec{S}_s\vec{D}_s^t \\
&= \vec{D}_s\vec{S}_s\vec{K}_s^t\vec{K}_s\vec{S}_s\vec{D}_s^t \\
&= \vec{D}_s\vec{S}_s\vec{S}_s\vec{D}_s^t \\
&= (\vec{D}_s\vec{S}_s)(\vec{D}_s\vec{S}_s)^t
\end{aligned}$$

In the above matrix, the (i, j) element quantifies the relationship between documents d_i and d_j.

To rank documents with regard to a given user query, we simply model the query as a *pseudo-document* in the original $\vec{M}$ term-document matrix. Assume the query is modeled as the document with number 0. Then, the first row in the matrix $\vec{M}_s^t\vec{M}_s$ provides the ranks of all documents with respect to this query.

Since the matrices used in the latent semantic indexing model are of rank s, $s << t$, and $s << N$, they form an efficient indexing scheme for the documents in the collection. Further, they provide for elimination of noise (present in index term-based representations) and removal of redundancy.

The latent semantic indexing model introduces an interesting conceptualization of the information retrieval problem based on the theory of singular value decomposition. Thus, it has its value as a new theoretical framework. Whether it is superior in practical situations with general collections remains to be verified.

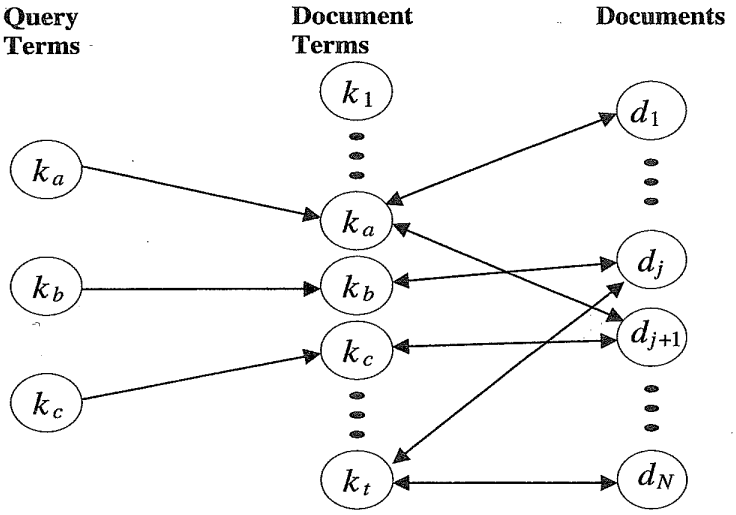


Figure 2.7 A neural network model for information retrieval.

2.7.3 Neural Network Model

In an information retrieval system, document vectors are compared with query vectors for the computation of a ranking. Thus, index terms in documents and queries have to be matched and weighted for computing this ranking. Since neural networks are known to be good pattern matchers, it is natural to consider their usage as an alternative model for information retrieval.

It is now well established that our brain is composed of billions of neurons. Each *neuron* can be viewed as a basic processing unit which, when stimulated by input signals, might emit output signals as a reactive action. The signals emitted by a neuron are fed into other neurons (through synaptic connections) which can themselves emit new output signals. This process might repeat itself through several layers of neurons and is usually referred to as a *spread activation* process. As a result, input information is processed (i.e., analyzed and interpreted) which might lead the brain to command physical reactions (e.g., motor actions) in response.

A *neural network* is an oversimplified graph representation of the mesh of interconnected neurons in a human brain. The nodes in this graph are the processing units while the edges play the role of the synaptic connections. To simulate the fact that the strength of a synaptic connection in the human brain changes over time, a *weight* is assigned to each edge of our neural network. At each instant, the state of a node is defined by its *activation level* (which is a function of its initial state and of the signals it receives as input). Depending on its activation level, a node A might send a signal to a neighbor node B. The strength of this signal at the node B depends on the weight associated with the edge between the nodes A and B.

A neural network for information retrieval can be defined as illustrated in Figure 2.7. The model depicted here is based on the work in [815]. We first

observe that the neural network in Figure 2.7 is composed of three layers: one for the query terms, one for the document terms, and a third one for the documents themselves. Observe the similarity between the topology of this neural network and the topology of the inference and belief networks depicted in Figures 2.9 and 2.10. Here, however, the query term nodes are the ones which initiate the inference process by sending signals to the document term nodes. Following that, the document term nodes might themselves generate signals to the document nodes. This completes a first phase in which a signal travels from the query term nodes to the document nodes (i.e., from the left to the right in Figure 2.7).

The neural network, however, does not stop after the first phase of signal propagation. In fact, the document nodes in their turn might generate new signals which are directed back to the document term nodes (this is the reason for the bidirectional edges between document term nodes and document nodes). Upon receiving this stimulus, the document term nodes might again fire new signals directed to the document nodes, repeating the process. The signals become weaker at each iteration and the spread activation process eventually halts. This process might activate a document d_l even when such a document does not contain any query terms. Thus, the whole process can be interpreted as the activation of a built-in thesaurus.

To the query term nodes is assigned an initial (and fixed) activation level equal to 1 (the maximum). The query term nodes then send signals to the document term nodes which are attenuated by normalized query term weights $\overline{w}_{i,q}$. For a vector-based ranking, these normalized weights can be derived from the weights $w_{i,q}$ defined for the vector model by equation 2.4. For instance,

$$\overline{w}_{i,q} = \frac{w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

where the normalization is done using the norm of the query vector.

Once the signals reach the document term nodes, these might send new signals out directed towards the document nodes. These signals are attenuated by normalized document term weights $\overline{w}_{i,j}$ derived from the weights $w_{i,j}$ defined for the vector model by equation 2.3. For instance,

$$\overline{w}_{i,j} = \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}}$$

where the normalization is done using the norm of the document vector.

The signals which reach a document node are summed up. Thus, after the first round of signal propagation, the activation level of the document node associated to the document d_j is given by

$$\sum_{i=1}^{t} \overline{w}_{i,q}\ \overline{w}_{i,j} = \frac{\sum_{i=1}^{t} w_{i,q}\ w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,j}^2}}$$

which is exactly the ranking provided by the classic vector model.

To improve the retrieval performance, the network continues with the spreading activation process after the first round of propagation. This modifies the initial vector ranking in a process analogous to a user relevance feedback cycle (see Chapter 5). To make the process more effective, a minimum activation threshold might be defined such that document nodes below this threshold send no signals out. Details can be found in [815].

There is no conclusive evidence that a neural network provides superior retrieval performance with general collections. In fact, the model has not been tested extensively with large document collections. However, a neural network does present an alternative modeling paradigm. Further, it naturally allows retrieving documents which are not initially related to the query terms — an appealing functionality.

2.8 Alternative Probabilistic Models

One alternative which has always been considered naturally appealing for quantifying document relevance is the usage of probability theory and its main streams. One such stream which is gaining increased attention concerns the *Bayesian belief networks* which we now discuss.

Bayesian (belief) networks are useful because they provide a clean formalism for combining distinct sources of evidence (past queries, past feedback cycles, and distinct query formulations) in support of the rank for a given document. This combination of distinct evidential sources can be used to improve retrieval performance (i.e., to improve the 'quality' of the ranked list of retrieved documents) as has been demonstrated in the work of Turtle and Croft [771].

In this chapter we discuss two models for information retrieval based on Bayesian networks. The first model is called *inference network* and provides the theoretical basis for the retrieval engine in the Inquery system [122]. Its success has attracted attention to the use of Bayesian networks with information retrieval systems. The second model is called *belief network* and generalizes the first model. At the end, we briefly compare the two models.

Our discussion below uses a style which is quite distinct from that employed by Turtle and Croft in their original writings. Particularly, we pay more attention to probabilistic argumentation during the development of the model. We make a conscious effort of consistently going back to the Bayesian formalism for motivating the major design decisions. It is our view that such an explanation strategy allows for a more precise argumentation which facilitates the task of grasping the subtleties involved.

Before proceeding, we briefly introduce Bayesian networks.

2.8.1 Bayesian Networks

Bayesian networks [630] are directed acyclic graphs (DAGs) in which the nodes represent random variables, the arcs portray causal relationships between these

variables, and the strengths of these causal influences are expressed by conditional probabilities. The *parents* of a node (which is then considered as a *child* node) are those judged to be direct *causes* for it. This causal relationship is represented in the DAG by a link directed from each parent node to the child node. The *roots* of the network are the nodes without parents.

Let x_i be a node in a Bayesian network G and Γ_{x_i} be the set of parent nodes of x_i. The influence of Γ_{x_i} on x_i can be specified by any set of functions $F_i(x_i, \Gamma_{x_i})$ that satisfy

$$\begin{aligned} \sum_{\forall x_i} F_i(x_i, \Gamma_{x_i}) &= 1 \\ 0 \leq F_i(x_i, \Gamma_{x_i}) &\leq 1 \end{aligned}$$

where x_i also refers to the states of the random variable associated to the node x_i. This specification is complete and consistent because the product $\prod_{\forall i} F_i(x_i, \Gamma_{x_i})$ constitutes a joint probability distribution for the nodes in G.

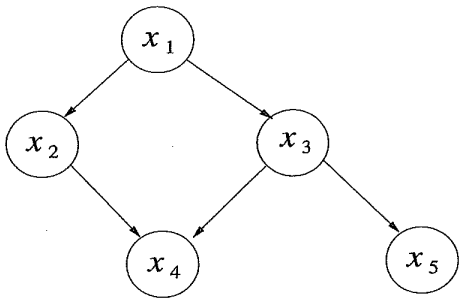


Figure 2.8 An example of a Bayesian network.

Figure 2.8 illustrates a Bayesian network for a joint probability distribution $P(x_1, x_2, x_3, x_4, x_5)$. In this case, the dependencies declared in the network allow the natural expression of the joint probability distribution in terms of local conditional probabilities (a key advantage of Bayesian networks) as follows.

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1)P(x_2|x_1)P(x_3|x_1)P(x_4|x_2, x_3)P(x_5|x_3)$$

The probability $P(x_1)$ is called the *prior* probability for the network and can be used to model previous knowledge about the semantics of the application.

2.8.2 Inference Network Model

The two most traditional schools of thought in probability are based on the *frequentist* view and the *epistemological* view. The frequentist view takes probability as a statistical notion related to the laws of chance. The epistemological

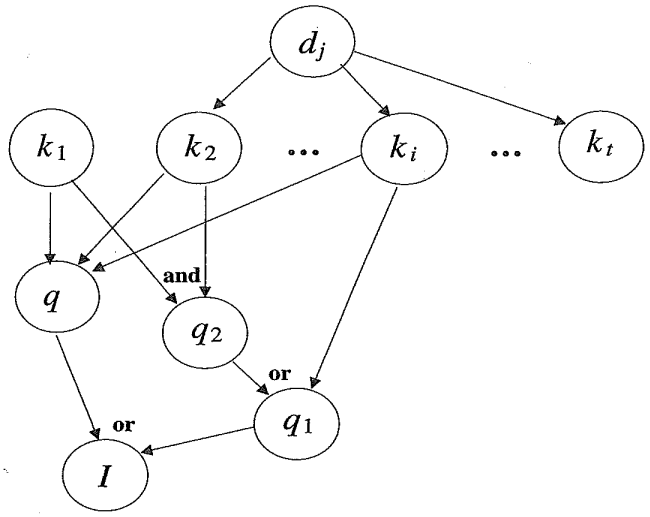


Figure 2.9 Basic inference network model.

view interprets probability as a degree of belief whose specification might be devoid of statistical experimentation. This second viewpoint is important because we frequently refer to probabilities in our daily lives without a clear definition of the statistical experiment which yielded those probabilities.

The inference network model [772, 771] takes an epistemological view of the information retrieval problem. It associates random variables with the index terms, the documents, and the user queries. A random variable associated with a document d_j represents the event of observing that document (i.e., the model assumes that documents are being observed in the search for relevant documents). The observation of the document d_j asserts a belief upon the random variables associated with its index terms. Thus, observation of a document is the *cause* for an increased belief in the variables associated with its index terms.

Index term and document variables are represented as nodes in the network. Edges are directed from a document node to its term nodes to indicate that observation of the document yields improved belief on its term nodes.

The random variable associated with the user query models the event that the information request specified by the query has been met. This random variable is also represented by a node in the network. The belief in this (query) node is a function of the beliefs in the nodes associated with the query terms. Thus, edges are directed from the index term nodes to the query node. Figure 2.9 illustrates an inference network for information retrieval. The document d_j has k_2, k_i, and k_t as its index terms. This is modeled by directing the edges from the node d_j to the nodes k_2, k_i, and k_t. The query q is composed of the index terms k_1, k_2, and k_i. This is modeled by directing the edges from the nodes k_1, k_2, and k_i to the node q. Notice that Figure 2.9 also includes three extra nodes: q_2, q_1, and I. The nodes q_2 and q_1 are used to model an (alternative) Boolean formulation q_1 for the query q (in this case, $q_1 = (k_1 \wedge k_2) \vee k_i$). When such

(additional) information is available, the user information need I is supported by both q and q_1.

In what follows, we concentrate our attention on the support provided to the query node q by the observation of a document d_j. Later on, we discuss the impact of considering multiple query representations for an information need I. This is important because, as Turtle and Croft have demonstrated, a keyword-based query formulation (such as q) can be combined with a Boolean-like query formulation (such as q_1) to yield improved retrieval performance for the same information need.

The complete inference network model also includes text nodes and query concept nodes but the model discussed above summarizes the essence of the approach.

A simplifying assumption is made which states that all random variables in the network are binary. This seems arbitrary but it does simplify the modeling task and is general enough to capture all the important relationships in the information retrieval problem.

Definition *Let $\vec{k}$ be a t-dimensional vector defined by $\vec{k} = (k_1, k_2, \ldots, k_t)$ where k_1, k_2, ..., k_t are binary random variables i.e., $k_i \in \{0, 1\}$. These variables define the 2^t possible states for $\vec{k}$. Further, let d_j be a binary random variable associated with a document d_j and let q be a binary random variable associated with the user query.*

Notice that q is used to refer to the query, to the random variable associated with it, and to the respective node in the network. This is also the case for d_j and for each index term k_i. We allow this overloading in syntax because it should always be clear whether we are referring to either the query or to its associated random variable.

The ranking of a document d_j with respect to a query q is a measure of how much evidential support the observation of d_j provides to the query q. In an inference network, the ranking of a document d_j is computed as $P(q \wedge d_j)$ where q and d_j are short representations for $q = 1$ and $d_j = 1$, respectively. In general, such a ranking is given by

$$\begin{aligned} P(q \wedge d_j) &= \sum_{\forall \vec{k}} P(q \wedge d_j | \vec{k}) \times P(\vec{k}) \\ &= \sum_{\forall \vec{k}} P(q \wedge d_j \wedge \vec{k}) \\ &= \sum_{\forall \vec{k}} P(q | d_j \times \vec{k}) \times P(d_j \times \vec{k}) \\ &= \sum_{\forall \vec{k}} P(q | \vec{k}) \times P(\vec{k} | d_j) \times P(d_j) \qquad \textbf{(2.6)} \\ P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j) \end{aligned}$$

which is obtained by basic conditioning and the application of Bayes' rule. Notice that $P(q|d_j \times \vec{k}) = P(q|\vec{k})$ because the k_i nodes separate the query node q from the document node d_j. Also, the notation $\overline{q \wedge d_j}$ is a short representation for $\neg(q \wedge d_j)$.

The instantiation of a document node d_j (i.e., the observation of the document) separates its children index term nodes making them mutually independent (see Bayesian theory for details). Thus, the degree of belief asserted to each index term node k_i by instantiating the document node d_j can be computed separately. This implies that $P(\vec{k}|d_j)$ can be computed in product form which yields (from equation 2.6),

$$\begin{aligned} P(q \wedge d_j) &= \sum_{\forall \vec{k}} P(q|\vec{k}) \times \\ &\left(\prod_{\forall i|g_i(\vec{k})=1} P(k_i|d_j) \times \prod_{\forall i|g_i(\vec{k})=0} P(\overline{k}_i|d_j) \right) \times P(d_j) \qquad (2.7) \\ P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j) \end{aligned}$$

where $P(\overline{k}_i|d_j) = 1 - P(k_i|d_j)$. Through proper specification of the probabilities $P(q|\vec{k})$, $P(k_i|d_j)$, and $P(d_j)$, we can make the inference network cover a wide range of useful information retrieval ranking strategies. Later on, we discuss how to use an inference network to subsume the Boolean model and tf-idf ranking schemes. Let us first cover the specification of the $P(d_j)$ probabilities.

Prior Probabilities for Inference Networks

Since the document nodes are the root nodes in an inference network, they receive a *prior probability distribution* which is of our choosing. This prior probability reflects the probability associated to the event of observing a given document d_j (to simplify matters, a single document node is observed at a time). Since we have no prior preferences for any document in particular, we usually adopt a prior probability distribution which is uniform. For instance, in the original work on inference networks [772, 771], the probability of observing a document d_j is set to $1/N$ where N is the total number of documents in the system. Thus,

$$\begin{aligned} P(d_j) &= \frac{1}{N} \\ P(\overline{d}_j) &= 1 - \frac{1}{N} \end{aligned}$$

The choice of the value $1/N$ for the prior probability $P(d_j)$ is a simple and natural specification given that our collection is composed of N documents. However, other specifications for $P(d_j)$ might also be interesting. For instance, in the cosine formula of the vector model, the contribution of an index term to

the rank of the document d_j is inversely proportional to the norm of the vector $\vec{d_j}$. The larger the norm of the document vector, the smaller is the relative contribution of its index terms to the document final rank. This effect can be taken into account through proper specification of the prior probabilities $P(d_j)$ as follows.

$$\begin{aligned} P(d_j) &= \frac{1}{|\vec{d_j}|} \\ P(\overline{d}_j) &= 1 - P(d_j) \end{aligned}$$

where $|\vec{d_j}|$ stands for the norm of the vector $\vec{d_j}$. Therefore, in this case, the larger the norm of a document vector, the smaller its associated prior probability. Such specification reflects a prior knowledge that we have about the behavior of vector-based ranking strategies (which normalize the ranking in the document space). As commanded by Bayesian postulates, previous knowledge of the application domain should be asserted in the specification of the priors in the network, as we have just done.

Inference Network for the Boolean Model

Here we demonstrate how an inference network can be tuned to subsume the Boolean model. First, for the Boolean model, the prior probabilities $P(d_j)$ are all set to $1/N$ because the Boolean model makes no prior distinction on documents. Thus,

$$\begin{aligned} P(d_j) &= \frac{1}{N} \\ P(\overline{d}_j) &= 1 - P(d_j) \end{aligned}$$

Regarding the conditional probabilities $P(k_i|d_j)$ and $P(q|\vec{k})$, the specification is as follows.

$$\begin{aligned} P(k_i|d_j) &= \begin{cases} 1 & \text{if } g_i(d_j) = 1 \\ 0 & \text{otherwise} \end{cases} \\ P(\overline{k}_i|d_j) &= 1 - P(k_i|d_j) \end{aligned}$$

which basically states that, when the document d_j is being observed, only the nodes associated with the index terms of the document d_j are *active* (i.e., have an induced probability greater than 0). For instance, observation of a document node d_j whose term vector is composed of exactly the index terms k_2, k_i, and k_t (see Figure 2.9) activates the index term nodes $\{k_2,k_i,k_t\}$ and no others.

Once the beliefs in the index term nodes have been computed, we can use them to compute the evidential support they provide to the user query q as

follows.

$$\begin{aligned} P(q|\vec{k}) &= \begin{cases} 1 & \text{if } \exists \vec{q}_{cc} \mid (\vec{q}_{cc} \in \vec{q}_{dnf}) \wedge (\forall_{k_i},\ g_i(\vec{k}) = g_i(\vec{q}_{cc})) \\ 0 & \text{otherwise} \end{cases} \\ P(\overline{q}|\vec{k}) &= 1 - P(q|\vec{k}) \end{aligned}$$

where $\vec{q}_{cc}$ and $\vec{q}_{dnf}$ are as defined for the classic Boolean model. The above equation basically states that one of the conjunctive components of the user query (expressed in disjunctive normal form) must be matched by the set of active terms in $\vec{k}$ (in this case, those activated by the document observed) exactly.

Substituting the above definitions for $P(q|\vec{k})$, $P(k_i|d_j)$, and $P(d_j)$ into equation 2.7, it can be easily shown that the set of documents retrieved is exactly the set of documents returned by the Boolean model as defined in section 2.5.2. Thus, an inference network can be used to subsume the Boolean model without difficulties.

Inference Network for tf-idf Ranking Strategies

For tf-idf ranking strategies (i.e., those related to the vector model), we adopt prior probabilities which reflect our prior knowledge of the importance of document normalization. Thus, we set the prior $P(d_j)$ to $1/|\vec{d}_j|$ as follows.

$$\begin{aligned} P(d_j) &= \frac{1}{|\vec{d}_j|} \\ P(\overline{d}_j) &= 1 - P(d_j) \end{aligned} \tag{2.8}$$

Further, we have to decide where to introduce the tf (term-frequency) and the idf (inverse-document-frequency) factors in the network. For that purpose, we consider that the tf and idf factors are normalized (as in equation 2.1) and that these normalized factors are strictly smaller than 1.

We first focus on capturing the impact of the tf factors in the network. Normalized tf factors are taken into account through the beliefs asserted upon the index term nodes as follows.

$$\begin{aligned} P(k_i|d_j) &= f_{i,j} \\ P(\overline{k}_i|d_j) &= 1 - P(k_i|d_j) \end{aligned} \tag{2.9}$$

These equations simply state that, according to the observed document d_j, the relevance of a term k_i is determined by its normalized term-frequency factor.

We are now in a position to consider the influence of idf factors. They are taken into account through the specification of the impact of index term nodes

on the query node. Define a vector $\vec{k}_i$ given by,

$$\vec{k}_i = \vec{k} \mid (g_i(\vec{k}) = 1 \wedge \forall_{j \neq i}\ g_j(\vec{k}) = 0)$$

The vector $\vec{k}_i$ is a reference to the state of the vector $\vec{k}$ in which the node k_i is active and all others are inactive. The motivation is that tf-idf ranking strategies sum up the *individual* contributions of index terms and $\vec{k}_i$ allows us to consider the influence of the term k_i in isolation. We are now ready to define the influence of the index term nodes in the query node q as

$$\begin{aligned} P(q|\vec{k}) &= \begin{cases} idf_i & \text{if } \vec{k} = \vec{k}_i \wedge g_i(\vec{q}) = 1 \\ 0 & \text{if } \vec{k} \neq \vec{k}_i \vee g_i(\vec{q}) = 0 \end{cases} \\ P(\overline{q}|\vec{k}) &= 1 - P(q|\vec{k}) \end{aligned} \tag{2.10}$$

where idf_i here is a normalized version of the idf factor defined in equation 2.2.

By applying equations 2.8, 2.9, and 2.10 to equation 2.7, we can then write

$$\begin{aligned} P(q \wedge d_j) &= \sum_{\forall \vec{k}_i} P(q|\vec{k}_i) \times P(k_i|d_j) \times \left(\prod_{\forall\ l \neq i} P(\overline{k}_l|d_j) \right) \times P(d_j) \\ &= \left(\prod_{\forall\ i} P(\overline{k}_i|d_j) \right) \times P(d_j) \times \sum_{\forall \vec{k}_i} P(k_i|d_j) \times P(q|\vec{k}_i) \times \frac{1}{P(\overline{k}_i|d_j)} \\ &= C_j \times \frac{1}{|\vec{d}_j|} \times \sum_{\forall i | g_i(\vec{d}_j)=1 \wedge g_i(\vec{q})=1} f_{i,j} \times idf_i \times \frac{1}{1 - f_{i,j}} \\ P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j) \end{aligned}$$

which provides a tf-idf-like ranking. Unfortunately, C_j depends on a product of the various probabilities $P(\overline{k}_i|d_j)$ which vary from document to document and thus the ranking is distinct from the one provided by the vector model. Despite this peculiarity in the tf-idf ranking generated, it has been shown that an inference network is able to provide good retrieval performance with general collections. The reason is that the network allows us to consistently combine evidence from distinct evidential sources to improve the final ranking, as we now discuss.

Combining Evidential Sources

In Figure 2.9, the first query node q is the standard keyword-based query formulation for the user information need I. The second query q_1 is a Boolean-like query formulation for the same information need (i.e., an additional evidential source collected from a specialist). The joint support these two query formulations provide to the information need node I can be modeled through, for

instance, an OR operator (i.e., $I = q \vee q_1$). In this case, the ranking provided by the inference network is computed as,

$$\begin{aligned} P(I \wedge d_j) &= \sum_{\vec{k}} P(I|\vec{k}) \times P(\vec{k}|d_j) \times P(d_j) \\ &= \sum_{\vec{k}} (1 - P(\overline{q}|\vec{k})\ P(\overline{q}_1|\vec{k})) \times P(\vec{k}|d_j) \times P(d_j) \end{aligned}$$

which might yield a retrieval performance which surpasses the retrieval performance obtained with each of the query nodes in isolation as demonstrated in [771].

2.8.3 Belief Network Model

The belief network model, introduced in 1996 by Ribeiro-Neto and Muntz [674], is also based on an epistemological interpretation of probabilities. However, it departs from the inference network model by adopting a clearly defined sample space. As a result, it yields a slightly different network topology which provides a separation between the document and the query portions of the network. This is the main difference between the two models and one which has theoretical implications.

The Probability Space

The probability space adopted was first introduced by Wong and Yao [830] and works as follows. All documents in the collection are indexed by *index terms* and the universe of discourse U is the set K of all index terms.

Definition *The set $K = \{k_1, \ldots, k_t\}$ is the universe of discourse and defines the sample space for the belief network model. Let $u \subset K$ be a subset of K. To each subset u is associated a vector $\vec{k}$ such that $g_i(\vec{k}) = 1 \Longleftrightarrow k_i \in u$.*

The introduction of the vector $\vec{k}$ is useful to keep the notation compatible with the one which has been used throughout this chapter.

Each index term is viewed as an *elementary concept* and K as a concept space. A concept u is a subset of K and might represent a document in the collection or a user query. In a belief network, set relationships are specified using random variables as follows.

Definition *To each index term k_i is associated a binary random variable which is also referred to as k_i. The random variable k_i is set to 1 to indicate that the index k_i is a member of a concept/set represented by $\vec{k}$.*

This association of concepts with subsets is useful because it allows us to express the logical notions of conjunction, disjunction, negation, and implication as the more familiar set-theoretic notions of intersection, union, complementation, and inclusion. Documents and user queries can be defined as concepts in the sample space K as follows.

Definition *A document d_j in the collection is represented as a concept (i.e., a set) composed of the terms which are used to index d_j. Analogously, a user query q is represented as a concept composed of the terms which are used to index q.*

A probability distribution P is defined over K as follows. Let c be a generic concept in the space K representing a document or user query. Then,

$$P(c) = \sum_u P(c|u) \times P(u) \tag{2.11}$$

$$P(u) = \left(\frac{1}{2}\right)^t \tag{2.12}$$

Equation 2.11 defines $P(c)$ as the *degree of coverage* of the space K by c. Such a coverage is computed by contrasting each of the concepts in K with c (through $P(c|u)$) and by summing up the individual contributions. This sum is weighted by the probability $P(u)$ with which u occurs in K. Since at the beginning the system has no knowledge of the probability with which a concept u occurs in the space K, we can assume that each u is equally likely which results in equation 2.12.

Belief Network Model

In the belief network model, the user query q is modeled as a network node to which is associated a binary random variable (as in the inference network model) which is also referred to as q. This variable is set to 1 whenever q completely covers the concept space K. Thus, when we assess $P(q)$ we compute the degree of coverage of the space K by q. This is equivalent to assessing the degree of belief associated with the following proposition: Is it true that q completely covers all possible concepts in K?

A document d_j is modeled as a network node with which is associated a binary random variable which is also referred to as d_j. This variable is 1 to indicate that d_j completely covers the concept space K. When we assess $P(d_j)$, we compute the degree of coverage of the space K by d_j. This is equivalent to assessing the degree of belief associated with the following proposition: Is it true that d_j completely covers all possible concepts in K?

According to the above formalism, the user query and the documents in the collection are modeled as subsets of index terms. Each of these subsets is interpreted as a *concept* embedded in the concept space K which works as a common *sample space*. Furthermore, user queries and documents are modeled

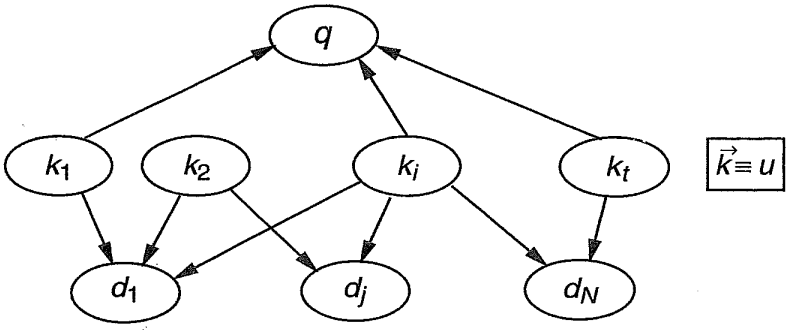


Figure 2.10 Basic belief network model.

identically. This is an important observation because it defines the topology of the belief network.

Figure 2.10 illustrates our belief network model. As in the inference network model, a query q is modeled as a binary random variable which is pointed to by the index term nodes which compose the query concept. Documents are treated analogously to user queries (i.e., both are concepts in the space K). Thus, contrary to the inference network model, a document node is pointed to by the index term nodes which compose the document. This is the topological difference between the two models and one which has more implications than it seems at first glance.

The ranking of a document d_j relative to a given query q is interpreted as a concept matching relationship and reflects the degree of coverage provided to the concept d_j by the concept q.

Assumption In the belief network model, $P(d_j|q)$ is adopted as the rank of the document d_j with respect to the query q.

By the application of Bayes' theorem, we can write $P(d_j|q) = P(d_j \wedge q)/P(q)$. Since $P(q)$ is a constant for all documents in the collection, we can write $P(d_j|q) \sim P(d_j \wedge q)$ i.e., the rank assigned to a document d_j is directly proportional to $P(d_j \wedge q)$. This last probability is computed through the application of equation 2.11 which yields

$$P(d_j|q) \sim \sum_{\forall u} P(d_j \wedge q|u) \times P(u)$$

In the belief network of Figure 2.10, instantiation of the index term variables logically separates the nodes q and d making them mutually independent (i.e., the document and query portions of the network are logically separated by in-

stantiation of the index term nodes). Therefore,

$$P(d_j|q) \sim \sum_{\forall u} P(d_j|u) \times P(q|u) \times P(u)$$

which can be rewritten as

$$P(d_j|q) \sim \sum_{\forall \vec{k}} P(d_j|\vec{k}) \times P(q|\vec{k}) \times P(\vec{k}) \quad \textbf{(2.13)}$$

To complete the belief network we need to specify the conditional probabilities $P(q|\vec{k})$ and $P(d_j|\vec{k})$. Distinct specifications of these probabilities allow the modeling of different ranking strategies (corresponding to different IR models). We now discuss how to specify these probabilities to subsume the vector model.

For the vector model, the probabilities $P(q|\vec{k})$ and $P(d_j|\vec{k})$ are specified as follows. Let,

$$\vec{k}_i = \vec{k} \mid (g_i(\vec{k}) = 1 \wedge \forall_{j \neq i}\ g_j(\vec{k}) = 0)$$

as before. Also,

$$P(q|\vec{k}) = \begin{cases} \frac{w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2}} & \text{if } \vec{k} = \vec{k}_i \wedge g_i(q) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{q}|\vec{k}) = 1 - P(q|\vec{k})$$

Further, define

$$P(d_j|\vec{k}) = \begin{cases} \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}} & \text{if } \vec{k} = \vec{k}_i \wedge g_i(\vec{d}_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{d_j}|\vec{k}) = 1 - P(d_j|\vec{k})$$

Then, the ordering of the retrieved documents (i.e., the ranking) defined by $P(d_j|q)$ coincides with the ordering generated by the vector model as specified in section 2.5.3. Thus, the belief network model can be tuned to subsume the vector model which cannot be accomplished with the inference network model.

2.8.4 Comparison of Bayesian Network Models

There is a close resemblance between the belief network model and the inference network model. However, this resemblance hides important differences between the two models. First, the belief network model is based on a set-theoretic view

of the IR ranking problem and adopts a clearly defined sample space. The inference network model takes a purely epistemological view of the IR problem which is more difficult to grasp (because, for instance, the sample space is not clearly defined). Second, the belief network model provides a separation between the document and the query portions of the network which facilitates the modeling of additional evidential sources such as past queries and past relevance information. Third, as a result of this document-query space separation, the belief network model is able to reproduce any ranking strategy generated by the inference network model while the converse is not true.

To see that the belief network ranking subsumes any ranking generated by an inference network, compare equations 2.6 and 2.13. The key distinction is between the terms $P(d_j|\vec{k})$ and $P(\vec{k}|d_j)$. For the latter, instantiation of the document node d_j separates the index term nodes making them mutually independent. Thus, the joint probability $P(\vec{k}|d_j)$ can always be computed as the product of the individual probabilities $P(k_i|d_j)$. However, the computation of $P(d_j|\vec{k})$ might be non-decomposable in a product of term-based probabilities. As a result, $P(d_j|\vec{k})$ can express any probability function defined with $P(\vec{k}|d_j)$ while the converse is not true.

One should not infer from the above comparison that the inference network model is not a good model. On the contrary, it has been shown in the literature that the inference network model allows top retrieval performance to be accomplished with general collections. Further, it is the retrieval model used by the Inquery system. The point of the comparison is that, from a theoretical point of view, the belief network model is more general. Also, it provides a separation between the document space and the query space which simplifies the modeling task.

2.8.5 Computational Costs of Bayesian Networks

In the inference network model, according to equation 2.6, only the states which have a single document active node are considered. Thus, the cost of computing the ranking is linear on the number of documents in the collection. As with conventional collections, index structures such as inverted files (see Chapter 8) are used to restrict the ranking computation to those documents which have terms in common with the query. Thus, the cost of computing an inference network ranking has the same complexity as the cost of computing a vectorial ranking.

In the belief network model, according to equation 2.13, the only states (of the roots nodes) considered (for computing the rank of a document d_j) are the ones in which the active nodes are exactly those associated with the query terms. Thus, again, the cost of computing the ranking is linear on the number of documents in the collection. If index structures are used, the cost of computing a belief network ranking has the same complexity as the cost of computing a vectorial ranking.

Therefore, the Bayesian network models discussed here do not impose significant additional costs for ranking computation. This is so because the networks presented do not include cycles, which implies that belief propagation can be done in a time proportional to the number of nodes in the network.

2.8.6 The Impact of Bayesian Network Models

The classic Boolean model is based on a neat formalism but is not very effective for information retrieval. The classic vector model provides improved answer sets but lacks a more formal framework. Many attempts have been made in the past to combine the best features of each model. The extended Boolean model and the generalized vector space model are two well known examples. These past attempts are grounded in the belief that the combination of selected properties from distinct models is a promising approach towards improved retrieval.

Bayesian network models constitute modern variants of probabilistic reasoning whose major strength (for information retrieval) is a framework which allows the neat combination of distinct evidential sources to support a relevance judgement (i.e., a numerical rank) on a given document. In this regard, belief networks seem more appropriate than previous approaches and more promising. Further, besides allowing the combination of Boolean and vector features, a belief network can be naturally extended to incorporate evidential information derived from past user sessions [674] and feedback cycles [332].

The inference network model has been successfully implemented in the Inquery retrieval system [122] and compares favorably with other retrieval systems. However, despite these promises, whether Bayesian networks will become popular and widely used for information retrieval remains to be seen.

2.9 Structured Text Retrieval Models

Consider a user with a superior visual memory. Such a user might then recall that the specific document he is interested in contains a page in which the string '*atomic holocaust*' appears in italic in the text surrounding a Figure whose label contains the word 'earth.' With a classic information retrieval model, this query could be expressed as ['atomic holocaust' and 'earth'] which retrieves all the documents containing both strings. Clearly, however, this answer contains many more documents than desired by this user. In this particular case, the user would like to express his query through a richer expression such as

same-page (near ('*atomic holocaust*,' Figure (label ('earth'))))

which conveys the details in his visual recollection. Further, the user might be interested in an advanced interface which simplifies the task of specifying this (now complex) query. This example illustrates the appeal of a query language which allows us to combine the specification of strings (or patterns) with the

specification of structural components of the document. Retrieval models which combine information on text content with information on the document structure are called *structured text retrieval* models.

For a query such as the one illustrated above, a structured text retrieval system searches for all the documents which *satisfy* the query. Thus, there is no notion of relevance attached to the retrieval task. In this sense, the current structured text retrieval models are data (instead of information) retrieval models. However, the retrieval system could search for documents which match the query conditions only partially. In this situation, the matching would be approximate and some ranking would have to be used for ordering the approximate answers. Thus, a structured text retrieval algorithm can be seen as an information retrieval algorithm for which the issue of appropriate ranking is not well established. In fact, this is an actual, interesting, and open research problem.

At the end of the 1980s and throughout the 1990s, various structured text retrieval models have appeared in the literature. Usually, the more expressive the model, the less efficient is its query evaluation strategy. Thus, selection of a structured model for a given application must be exercised with care. A good policy is to select the most efficient model which supports the functionality required by the application in view.

Here, we do not survey all the structured text retrieval models. Instead, we briefly discuss the main features of two of them, namely, a model based on *non-overlapping lists* and a model based on *proximal nodes*. These two models should provide a good overview of the main issues and tradeoffs in structured text retrieval.

We use the term *match point* to refer to the position in the text of a sequence of words which matches (or satisfies) the user query. Thus, if the user specifies the simple query ['atomic holocaust in Hiroshima'] and this string appears in three positions in the text of a document d_j, we say that the document d_j contains three match points. Further, we use the term *region* to refer to a contiguous portion of the text and the term *node* to refer to a structural component of the document such as a chapter, a section, a subsection, etc. Thus, a node is a region with predefined topological properties which are known both to the author of the document and to the user who searches the document system.

2.9.1 Model Based on Non-Overlapping Lists

Burkowski [132, 133] proposes to divide the whole text of each document in non-overlapping text regions which are collected in a *list*. Since there are multiple ways to divide a text in non-overlapping regions, multiple lists are generated. For instance, we might have a list of all chapters in the document, a second list of all sections in the document, and a third list of all subsections in the document. These lists are kept as separate and distinct data structures. While the text regions in the same (flat) list have no overlapping, text regions from distinct lists might overlap. Figure 2.11 illustrates four separate lists for the same document.

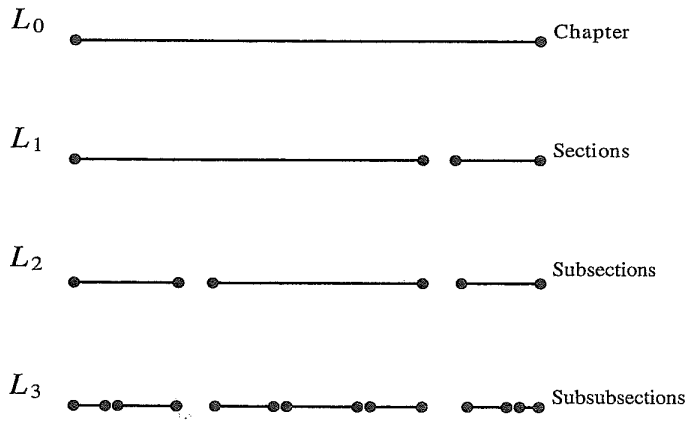


Figure 2.11 Representation of the structure in the text of a document through four separate (flat) indexing lists.

To allow searching for index terms and for text regions, a single inverted file (see Chapter 8 for a definition of inverted files) is built in which each structural component stands as an entry in the index. Associated with each entry, there is a list of text regions as a list of occurrences. Moreover, such a list could be easily merged with the traditional inverted file for the words in the text. Since the text regions are non-overlapping, the types of queries which can be asked are simple: (a) select a region which contains a given word (and does not contain other regions); (b) select a region A which does not contain any other region B (where B belongs to a list distinct from the list for A); (c) select a region not contained within any other region, etc.

2.9.2 Model Based on Proximal Nodes

Navarro and Baeza-Yates [41, 589, 590] propose a model which allows the definition of independent hierarchical (non-flat) indexing structures over the same document text. Each of these indexing structures is a strict hierarchy composed of chapters, sections, paragraphs, pages, and lines which are called *nodes* (see Figure 2.12). To each of these nodes is associated a text region. Further, two distinct hierarchies might refer to overlapping text regions.

Given a user query which refers to distinct hierarchies, the compiled answer is formed by nodes which all come from only one of them. Thus, an answer cannot be composed of nodes which come from two distinct hierarchies (which allows for faster query processing at the expense of less expressiveness). Notice, however, that due to the hierarchical structure, nested text regions (coming from the same hierarchy) are allowed in the answer set.

Figure 2.12 illustrates a hierarchical indexing structure composed of four

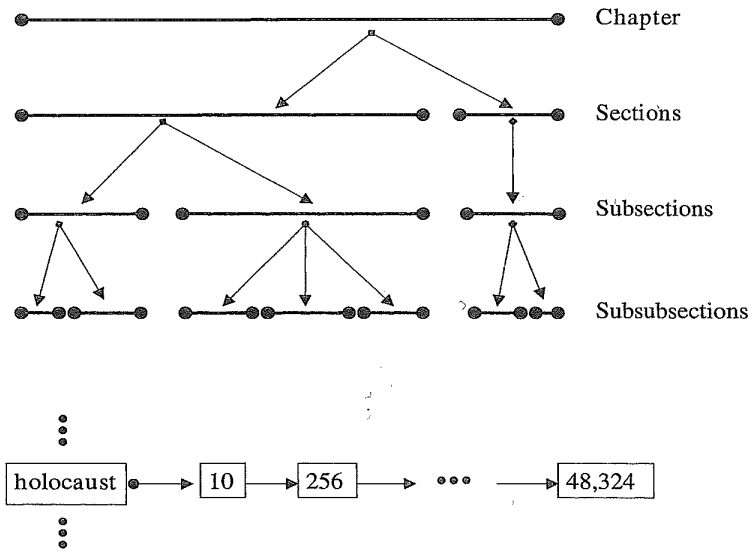


Figure 2.12 Hierarchical index for structural components and flat index for words.

levels (corresponding to a chapter, sections, subsections, and subsubsections of the same document) and an inverted list for the word 'holocaust.' The entries in this inverted list indicate all the positions in the text of the document in which the word 'holocaust' occurs. In the hierarchy, each node indicates the position in the text of its associated structural component (chapter, section, subsection, or subsubsection).

The query language allows the specification of regular expressions (to search for strings), the reference to structural components by name (to search for chapters, for instance), and a combination of these. In this sense, the model can be viewed as a compromise between expressiveness and efficiency. The somewhat limited expressiveness of the query language allows efficient query processing by first searching for the components which match the strings specified in the query and, subsequently, evaluating which of these components satisfy the structural part of the query.

Consider, for instance, the query [(*section) with ('holocaust')] which searches for sections, subsections, or subsubsections which contain the word 'holocaust.' A simple query processing strategy is to traverse the inverted list for the term 'holocaust' and, for each entry in the list (which indicates an occurrence of the term 'holocaust' in the text), search the hierarchical index looking for sections, subsections, and subsubsections containing that occurrence of the term. A more sophisticated query processing strategy is as follows. For the first entry in the list for 'holocaust,' search the hierarchical index as before. This implies traversing down the hierarchy until no more successful matches occur (or the bottom of the hierarchy is reached). Let the last matching structural component be referred to as the innermost matching component. Once this first search is concluded, do not start all over again for the following entry in the

inverted list. Instead, verify whether the innermost matching component also matches the second entry in the list. If it does, we immediately conclude that the larger structural components above it (in the hierarchy) also do. Proceed then to the third entry in the list, and so on. Notice that the query processing is accelerated because only the nearby (or proximal) nodes in the list need to be searched at each time. This is the reason for the label *proximal nodes.*

The model based on proximal nodes allows us to formulate queries which are more complex than those which can be formulated in the model based on non-overlapping lists. To speed up query processing, however, only nearby (proximal) nodes are looked at which imposes restrictions on the answer set retrieved (all nodes must come from the same hierarchy). More complex models for structured retrieval have been proposed in the literature as discussed in [41, 590].

2.10 Models for Browsing

As already observed, the user might not be interested in posing a specific query to the system. Instead, he might be willing to invest some time in exploring the document space looking for interesting references. In this situation, we say that the user is *browsing* the space instead of searching. Both with browsing and searching, the user has goals which he is pursuing. However, in general, the goal of a searching task is clearer in the mind of the user than the goal of a browsing task. As is obvious, this is not a distinction which is valid in all scenarios. But, since it is simple and provides a clear separation between the tasks of searching and browsing, it is adopted here. We distinguish three types of browsing namely, flat, structure guided, and hypertext.

2.10.1 Flat Browsing

The idea here is that the user explores a document space which has a flat organization. For instance, the documents might be represented as dots in a (two-dimensional) plan or as elements in a (single dimension) list. The user then glances here and there looking for information within the documents visited. For instance, he might look for correlations among neighbor documents or for keywords which are of interest to him. Such keywords could then be added to the original query in an attempt to provide better contextualization. This is a process called *relevance feedback* which is discussed in detail in Chapter 5. Also, the user could explore a single document in a flat manner. For example, he could use a browser to look into a Web page, using the arrows and the scroll bar. One disadvantage is that in a given page or screen there may not be any indication about the context where the user is. For example, if he opens a novel at a random page, he might not know in which chapter that page is.

Web search engines such as 'Yahoo!' provide, besides the standard search interface, a hierarchical directory which can be used for browsing (and frequently, for searching). However, the organization is not flat as discussed below.

2.10.2 Structure Guided Browsing

To facilitate the task of browsing, the documents might be organized in a structure such as a directory. Directories are hierarchies of classes which group documents covering related topics. Such hierarchies of classes have been used to classify document collections for many centuries now. Thus, it seems natural to adapt them for use with modern browsing interfaces. In this case, we say that the user performs a structure guided type of browsing. The same idea can be applied to a single document. For example, if we are browsing an electronic book, a first level of content could be the chapters, the second level, all sections, and so on. The last level would be the text itself (flat). A good user interface could go down or up those levels in a focused manner, assisting the user with the task of keeping track of the context.

Besides the structure which directs the browsing task, the interface can also include facilities such as a history map which identifies classes recently visited. This might be quite useful for dealing with very large structures – an issue discussed in Chapters 10 and 13. When searching, the occurrences can also be displayed showing just the structure (for example, using the table of contents). This allows us to see the occurrences in a global context instead of in a page of text that may have no indication of where we are.

2.10.3 The Hypertext Model

One fundamental concept related to the task of writing down text is the notion of *sequencing*. Written text is usually conceived to be read sequentially. The reader should not expect to fully understand the message conveyed by the writer by randomly reading pieces of text here and there. One might rely on the text structure to skip portions of the text but this might result in miscommunication between reader and writer. Thus, a sequenced organizational structure lies underneath most written text. When the reader fails to perceive such a structure and abide by it, he frequently is unable to capture the essence of the writer's message.

Sometimes, however, we are looking for information which is subsumed by the whole text but which cannot be easily captured through sequential reading. For instance, while glancing at a book about the history of the wars fought by man, we might be momentarily interested solely in the regional wars in Europe. We know that this information is in the book, but we might have a hard time finding it because the writer did not organize his writings with this purpose (he might have organized the wars chronologically). In such a situation, a different organization of the text is desired. However, there is no point in rewriting the whole text. Thus, the solution is to define a new organizational structure besides the one already in existence. One way to accomplish such a goal is through the design of a hypertext.

Hypertext Definition and the Navigational Task

A *hypertext* is a high level interactive navigational structure which allows us to browse text non-sequentially on a computer screen. It consists basically of nodes which are correlated by directed links in a graph structure.

To each node is associated a text region which might be a chapter in a book, a section in an article, or a Web page. Two nodes A and B might be connected by a *directed link* l_{AB} which correlates the texts associated with these two nodes. In this case, the reader might move to the node B while reading the text associated with the node A.

In its most conventional form, a hypertext link l_{AB} is attached to a specific string inside the text for node A. Such a string is marked specially (for instance, its characters might appear in a different color or underlined) to indicate the presence of the underlying link. While reading the text, the user might come across a marked string. If the user clicks on that string, the underlying directed link is followed, and a new text region (associated with the node at the destination) is displayed on the screen.

The process of navigating the hypertext can be understood as a traversal of a directed graph. The linked nodes of the graph represent text nodes which are semantically related. While traversing this graph the reader visualizes a flow of information which was conceived by the designer of the hypertext. Consider our previous example regarding a book on the wars fought by man. One might design a hypertext composed of two distinct *webs* (here, a web is simply a connected component formed by a subset of all links in the hypertext). While the first web might be designed to provide access to the local wars fought in Europe in chronological order, the second web might be designed to provide access to the local wars fought by each European country. In this way, the user of this hypertext can access the information according to his particular need.

When the hypertext is large, the user might lose track of the organizational structure of the hypertext. The effect is that the user might start to take bad navigational decisions which might sidetrack him from his main goal (which usually consists of finding a piece of information in the hypertext). When this happens, the user is said to be *lost in hyperspace* [604]. To avoid this problem, it is desirable that the hypertext include a hypertext map which shows where the user is at all times. In its simplest form, this map is a directed graph which displays the current node being visited. Additionally, such a map could include information on the paths the user has traveled so far. This can be used to remind the user of the uselessness of following paths which have been explored already.

While navigating a hypertext, the user is restricted to the intended flow of information previously conceived by the hypertext designer. Thus, the task of designing a hypertext should take into account the needs of its potential users. This implies the execution of a requirement analysis phase before starting the actual implementation of the hypertext. Such a requirement analysis is critically important but is frequently overlooked.

Furthermore, during the hypertext navigation, the user might find it difficult to orient himself. This difficulty arises even in the presence of a guiding

tool such as the hypertext map discussed above. One possible reason is an excessively complex hypertext organization with too many links which allow the user to travel back and forth. To avoid this problem, the hypertext can have a simpler structure which can be quickly remembered by the user at all times. For instance, the hypertext can be organized hierarchically to facilitate the navigational task.

Definition of the structure of the hypertext should be accomplished in a domain modeling phase (done after a requirement analysis phase). Further, after the modeling of the domain, a user interface design should be concluded prior to implementation. Only then, can we say that we have a proper hypertext structure for the application at hand. In the Web, however, pages are usually implemented with no attention paid to requirement analysis, domain modeling, and user interface design. As a result, Web pages are frequently poorly conceived and often fail to provide the user with a proper hypertext structure for assistance with the information seeking task.

With large hypertexts, it might be difficult for the user to position himself in the part of the whole graph which is of most interest to him. To facilitate this initial positioning step, a search based on index terms might be used. In [540], Manber discusses the advantages of this approach.

Hypertexts provided the basis for the conception and design of the hypertext markup language (HTML) and the hypertext transfer protocol (HTTP) which originated the *World Wide Web* (which we simply refer to as the Web). In Chapter 13, we discuss the Web in detail. We briefly discuss some of its features below.

About the Web

When one talks about the Web, the first concept which comes to mind is that of a hypertext. In fact, we frequently think of the Web as a huge distributed hypertext domain. However, the Web is not exactly a proper hypertext because it lacks an underlying data model, it lacks a navigational plan, and it lacks a consistently designed user interface. Each one of the millions of Web page designers devises his own interface with its own peculiar characteristics. Many times we visit a Web site simply looking for a phone number and cannot find it because it is buried in the least expected place of the local hypertext structure. Thus, the Web user has no underlying metaphor to assist him in the search for information of interest.

Instead of saying that the Web is a hypertext, we prefer to say that it is a pool of (partially) interconnected webs. Some of these webs might be characterized as a local hypertext (in the sense that they have an underlying structure which enjoys some consistency) but others might be simply a collection of pages designed separately (for instance, the web of a university department whose professors design their own pages). Despite not being exactly a hypertext, the Web has provided us with a new dimension in communication functionality because it is easily accessible world wide at very low cost. And maybe most important, the Web has no control body setting up regulations and censorship rules. As a

result, for the first time in the history of mankind, any one person can publish his writings through a large medium without being subjected to the filtering of an editorial board.

For a more thorough discussion of these and many other issues related to the Web, the user is referred to Chapter 13.

2.11 Trends and Research Issues

There are three main types of products and systems which can benefit directly from research in models for information retrieval: library systems, specialized retrieval systems, and the Web.

Regarding library systems, there is currently much interest in cognitive and behavioral issues oriented particularly at a better understanding of which criteria the users adopt to judge relevance. From the point of view of the computer scientist, a main question is how this knowledge about the user affects the ranking strategies and the user interface implemented by the system. A related issue is the investigation of how models other than the Boolean model (which is still largely adopted by most large commercial library systems) affect the user of a library.

A specialized retrieval system is one which is developed with a particular application in mind. For instance, the LEXIS-NEXIS retrieval system (see Chapter 14), which provides access to a very large collection of legal and business documents, is a good example of a specialized retrieval system. In such a system, a key problem is how to retrieve (almost) all documents which might be relevant to the user information need without also retrieving a large number of unrelated documents. In this context, sophisticated ranking algorithms are highly desirable. Since ranking based on single evidential sources is unlikely to provide the appropriate answers, research on approaches for combining several evidential sources seems highly relevant (as demonstrated at the various TREC conferences, see Chapter 3 for details).

In the Web, the scenario is quite distinct and unique. In fact, the user of the Web frequently does not know what he wants or has great difficulty in properly formulating his request. Thus, research in advanced user interfaces is highly desirable. From the point of view of the ranking engine, an interesting problem is to study how the paradigm adopted for the user interface affects the ranking. Furthermore, it is now well established that the indexes maintained by the various Web search engines are almost disjoint (e.g., the ten most popular search engines have indexes whose intersection corresponds to less than 2% of the total number of pages indexed). In this scenario, research on meta-search engines (i.e., engines which work by fusing the rankings generated by other search engines) seems highly promising.

2.12 Bibliographic Discussion

Early in 1960, Maron and Kuhns [547] had already discussed the issues of relevance and probabilistic indexing in information retrieval. Twenty-three years

later, Salton and McGill wrote a book [698] which became a classic in the field. The book provides a thorough coverage of the three classic models in information retrieval namely, the Boolean, the vector, and the probabilistic models. Another landmark reference is the book by van Rijsbergen [785] which, besides also covering the three classic models, presents a thorough and enjoyable discussion on the probabilistic model. The book edited by Frakes and Baeza-Yates [275] presents several data structures and algorithms for IR and is more recent. Further, it includes a discussion of ranking algorithms by Harman [340] which provides interesting insights into the history of information retrieval from 1960 to 1990.

Boolean operations and their implementation are covered in [803]. The inadequacy of Boolean queries for information retrieval was characterized early on by Verhoeff, Goffman, and Belzer [786]. The issue of adapting the Boolean formalism to operate with other frameworks received great attention. Bookstein discusses the problems related with merging Boolean and weighted retrieval systems [101] and the implications of Boolean structure for probabilistic retrieval [103]. Losee and Bookstein [522] cover the usage of Boolean queries with probabilistic retrieval. Anick *et al.* [21] propose an interface based on natural language for Boolean retrieval. A thesaurus-based Boolean retrieval system is proposed in [493].

The vector model is maybe the most popular model among the research community in information retrieval. Much of this popularity is due to the long-term research of Salton and his associates [697, 704]. Most of this research revolved around the SMART retrieval system developed at Cornell University [695, 842, 696]. Term weighting for the vector model has also been investigated thoroughly. Simple term weighting was used early on by Salton and Lesk [697]. Sparck Jones introduced the idf factor [409, 410] and Salton and Yang verified its effectiveness for improving retrieval [704]. Yu and Salton [842] further studied the effects of term weighting in the final ranking. Salton and Buckley [696] summarize 20 years of experiments in term weighting with the SMART system. Raghavan and Wong [665] provide a critical analysis of the vector model.

The probabilistic model was introduced by Robertson and Sparck Jones [677] and is thoroughly discussed in [785]. Experimental studies with the model were conducted by Sparck Jones [411, 412] which used feedback from the user to estimate the initial probabilities. Croft and Harper [199] proposed a method to estimate these probabilities without feedback from the user. Croft [198] later on added within-document frequency weights into the model. Fuhr discusses probabilistic indexing through polynomial retrieval functions [281, 284]. Cooper, Gey, and Dabney [186] and later on Gey [295] propose the use of logistic regression with probabilistic retrieval. Lee and Kantor [494] study the effect of inconsistent expert judgements on probabilistic retrieval. Fuhr [282] reviews various variants of the classic probabilistic model. Cooper [187], in a seminal paper, raises troubling questions on the utilization of the probabilistic ranking principle in information retrieval.

The inference network model was introduced by Turtle and Croft [772, 771] in 1990. Haines and Croft [332] discuss the utilization of inference networks for user relevance feedback (see Chapter 5). Callan, Lu, and Croft [139] use an

inference network to search distributed document collections. Callan [138], in his turn, discusses the application of inference networks to information filtering. The belief network model, due to Ribeiro-Neto and Muntz [674], generalizes the inference network model.

The extended Boolean model was introduced by Salton, Fox, and Wu [703]. Lee, Kim, Kim, and Lee [496] discuss the evaluation of Boolean operators with the extended Boolean model, while properties of the model are discussed in [495]. The generalized vector space model was introduced in 1985 by Wong, Ziarko, and Wong [832, 831]. Latent semantic indexing was introduced in 1988 by Furnas, Deerwester, Dumais, Landauer, Harshman, Streeter, and Lochbaum [287]. In a subsequent paper, Bartell, Cottrell, and Belew [62] show that latent semantic indexing can be interpreted as a special case of multidimensional scaling.

Regarding neural network models for information retrieval, our discussion in this book is based mainly on the work by Wilkinson and Hingston [815]. But we also benefited from the writings of Kwok on the subject and related topics [466, 467, 469, 468].

The fuzzy set model (for information retrieval) covered in this book is due to Ogawa, Morita, and Kobayashi [616]. The utilization of fuzzy theory in information retrieval goes back to the 1970s with the work of Radecki [658, 659, 660, 661], of Sachs [691], and of Tahani [755]. Bookstein [102] proposes the utilization of fuzzy operators to deal with weighted Boolean searches. Kraft and Buel [461] utilize fuzzy sets to generalize a Boolean system. Miyamoto, Miyake, and Nakayama [567] discuss the generation of a pseudothesaurus using co-occurrences and fuzzy operators. Subsequently, Miyamoto and Nakayama [568] discuss the utilization of this thesaurus with information retrieval systems.

Our discussion on structured text is based on the survey by [41]. Another survey of interest (an older one though) is the work by MacLeod [533]. Burkowski [132, 133] proposed a model based on non-overlapping regions. Clarke, Cormack, and Burkowski [173] extended this model with overlapping capabilities. The model based on proximal nodes was proposed by Navarro and Baeza-Yates [589, 590]. In [534], MacLeod introduced a model based on a single hierarchy which also associates attributes with nodes in the hierarchy (for database-like querying) and hypertext links with pairs of nodes. Kilpelainen and Mannila [439] discuss the retrieval from hierarchical texts through the specification of partial patterns. In [183], Consens and Milo discuss algebras for querying text regions.

A classic reference on hypertexts is the book by Nielsen [604]. Another popular reference is the book by Shneiderman and Kearsley [727]. Conklin [181] presents an introductory survey of the area. The *Communications of the ACM* dedicated an special edition [177] to hypermedia which discusses in detail the Dexter model — a reference standard on the terminology and semantics of basic hypermedia concepts. A subsequent edition [178] was dedicated to the presentation of various models for supporting the design of hypermedia applications.

Chapter 4
Query Languages

with Gonzalo Navarro

4.1 Introduction

We cover in this chapter the different kinds of queries normally posed to text retrieval systems. This is in part dependent on the retrieval model the system adopts, i.e., a full-text system will not answer the same kinds of queries as those answered by a system based on keyword ranking (as Web search engines) or on a hypertext model. In Chapter 8 we explain *how* the user queries are solved, while in this chapter we show *which* queries can be formulated. The type of query the user might formulate is largely dependent on the underlying information retrieval model. The different models for text retrieval systems are covered in Chapter 2.

As in previous chapters, we want to distinguish between information retrieval and data retrieval, as we use this dichotomy to classify different query languages. We have chosen to distinguish first languages that allow the answer to be ranked, that is, languages for information retrieval. As covered in Chapter 2, for the basic information retrieval models, keyword-based retrieval is the main type of querying task. For query languages not aimed at information retrieval, the concept of ranking cannot be easily defined, so we consider them as languages for data retrieval. Furthermore, some query languages are not intended for final users and can be viewed as languages that a higher level software package should use to query an on-line database or a CD-ROM archive. In that case, we talk about *protocols* rather than query languages. Depending on the user experience, a different query language will be used. For example, if the user knows exactly what he wants, the retrieval task is easier and ranking may not even be needed.

An important issue is that most query languages try to use the content (i.e., the semantics) and the structure of the text (i.e., the text syntax) to find relevant documents. In that sense, the system may fail to find the relevant answers (see Chapter 3). For this reason, a number of techniques meant to enhance the usefulness of the queries exist. Examples include the expansion of a word to the set of its synonyms or the use of a thesaurus and stemming to

put together all the derivatives of the same word. Moreover, some words which are very frequent and do not carry meaning (such as 'the'), called *stopwords*, may be removed. This subject is covered in Chapter 7. Here we assume that all the query preprocessing has already been done. Although these operations are usually done for information retrieval, many of them can also be useful in a data retrieval context. When we want to emphasize the difference between words that can be retrieved by a query and those which cannot, we call the former 'keywords.'

Orthogonal to the kind of queries that can be asked is the subject of the *retrieval unit* the information system adopts. The retrieval unit is the basic element which can be retrieved as an answer to a query (normally a set of such basic elements is retrieved, sometimes ranked by relevance or other criterion). The retrieval unit can be a file, a document, a Web page, a paragraph, or some other structural unit which contains an answer to the search query. From this point on, we will simply call those retrieval units 'documents,' although as explained this can have different meanings (see also Chapter 2).

This chapter is organized as follows. We first show the queries that can be formulated with keyword-based query languages. They are aimed at information retrieval, including simple words and phrases as well as Boolean operators which manipulate sets of documents. In the second section we cover pattern matching, which includes more complex queries and is generally aimed at complementing keyword searching with more powerful data retrieval capabilities. Third, we cover querying on the structure of the text, which is more dependent on the particular text model. Finally, we finish with some standard protocols used on the Internet and by CD-ROM publishers.

4.2 Keyword-Based Querying

A query is the formulation of a user information need. In its simplest form, a query is composed of keywords and the documents containing such keywords are searched for. Keyword-based queries are popular because they are intuitive, easy to express, and allow for fast ranking. Thus, a query can be (and in many cases is) simply a word, although it can in general be a more complex combination of operations involving several words.

In the rest of this chapter we will refer to single-word and multiple-word queries as *basic queries*. Patterns, which are covered in section 4.3, are also considered as basic queries.

4.2.1 Single-Word Queries

The most elementary query that can be formulated in a text retrieval system is a word. Text documents are assumed to be essentially long sequences of words. Although some models present a more general view, virtually all models allow us

to see the text in this perspective and to search words. Some models are also able to see the internal division of words into letters. These latter models permit the searching of other types of patterns, which are covered in section 4.3. The set of words retrieved by these extended queries can then be fed into the word-treating machinery, say to perform thesaurus expansion or for ranking purposes.

A word is normally defined in a rather simple way. The alphabet is split into 'letters' and 'separators,' and a word is a sequence of letters surrounded by separators. More complex models allow us to specify that some characters are not letters but do not split a word, e.g. the hyphen in '`on-line`.' It is good practice to leave the choice of what is a letter and what is a separator to the manager of the text database.

The division of the text into words is not arbitrary, since words carry a lot of meaning in natural language. Because of that, many models (such as the vector model) are completely structured on the concept of words, and words are the only type of queries allowed (moreover, some systems only allow a small set of words to be extracted from the documents). The result of word queries is the set of documents containing at least one of the words of the query. Further, the resulting documents are ranked according to a degree of similarity to the query. To support ranking, two common statistics on word occurrences inside texts are commonly used: 'term frequency' which counts the number of times a word appears inside a document and 'inverse document frequency' which counts the number of documents in which a word appears. See Chapter 2 for more details.

Additionally, the exact positions where a word appears in the text may be required for instance, by an interface which highlights each occurrence of that word.

4.2.2 Context Queries

Many systems complement single-word queries with the ability to search words in a given *context*, that is, near other words. Words which appear near each other may signal a higher likelihood of relevance than if they appear apart. For instance, we may want to form phrases of words or find words which are proximal in the text. Therefore, we distinguish two types of queries:

- **Phrase** is a sequence of single-word queries. An occurrence of the phrase is a sequence of words. For instance, it is possible to search for the word '`enhance`,' and then for the word '`retrieval`.' In phrase queries it is normally understood that the separators in the text need not be the same as those in the query (e.g., two spaces versus one space), and uninteresting words are not considered at all. For instance, the previous example could match a text such as '...enhance the retrieval...'. Although the notion of a phrase is a very useful feature in most cases, not all systems implement it.
- **Proximity** A more relaxed version of the phrase query is the proximity query. In this case, a sequence of single words or phrases is given, together

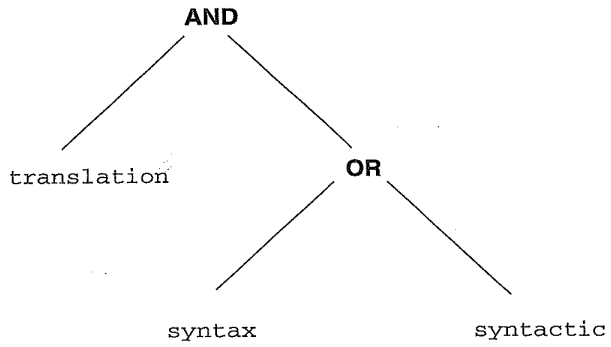


Figure 4.1 An example of a query syntax tree. It will retrieve all the documents which contain the word '`translation`' as well as either the word '`syntax`' or the word '`syntactic`'.

with a maximum allowed distance between them. For instance, the above example could state that the two words should occur within four words, and therefore a match could be '`...enhance the power of retrieval...`' This distance can be measured in characters or words depending on the system. The words and phrases may or may not be required to appear in the same order as in the query.

Phrases can be ranked in a fashion somewhat analogous to single words (see Chapters 2 and 5 for details). Proximity queries can be ranked in the same way if the parameters used by the ranking technique do not depend on physical proximity. Although it is not clear how to do better ranking, physical proximity has semantic value. This is because in most cases the proximity means that the words are in the same paragraph, and hence related in some way.

4.2.3 Boolean Queries

The oldest (and still heavily used) form of combining keyword queries is to use Boolean operators. A *Boolean query* has a syntax composed of *atoms* (i.e., basic queries) that retrieve documents, and of *Boolean operators* which work on their operands (which are sets of documents) and deliver sets of documents. Since this scheme is in general *compositional* (i.e., operators can be composed over the results of other operators), a *query syntax tree* is naturally defined, where the leaves correspond to the basic queries and the internal nodes to the operators. The query syntax tree operates on an algebra over sets of documents (and the final answer of the query is also a set of documents). This is much as, for instance, the syntax trees of arithmetic expressions where the numbers and variables are the leaves and the operations form the internal nodes. Figure 4.1 shows an example.

The operators most commonly used, given two basic queries or Boolean

subexpressions e_1 and e_2, are:

- **OR** The query (e_1 OR e_2) selects all documents which satisfy e_1 or e_2. Duplicates are eliminated.
- **AND** The query (e_1 AND e_2) selects all documents which satisfy both e_1 and e_2.
- **BUT** The query (e_1 BUT e_2) selects all documents which satisfy e_1 but not e_2. Notice that classical Boolean logic uses a NOT operation, where (NOT e_2) is valid whenever e_2 is not. In this case all documents not satisfying e_2 should be delivered, which may retrieve a huge amount of text and is probably not what the user wants. The BUT operator, instead, restricts the universe of retrievable elements to the result of e_1.†

Besides selecting the appropriate documents, the IR system may also sort the documents by some criterion, highlight the occurrences within the documents of the words mentioned in the query, and allow feedback by taking the answer set as a basis to reformulate the query.

With classic Boolean systems, no ranking of the retrieved documents is normally provided. A document either satisfies the Boolean query (in which case it is retrieved) or it does not (in which case it is not retrieved). This is quite a limitation because it does not allow for partial matching between a document and a user query. To overcome this limitation, the condition for retrieval must be relaxed. For instance, a document which partially satisfies an AND condition might be retrieved.

In fact, it is widely accepted that users not trained in mathematics find the meaning of Boolean operators difficult to grasp. With this problem in mind, a 'fuzzy Boolean' set of operators has been proposed. The idea is that the meaning of AND and OR can be relaxed, such that instead of forcing an element to appear in *all* the operands (AND) or at least in *one* of the operands (OR), they retrieve elements appearing in *some* operands (the AND may require it to appear in more operands than the OR). Moreover, the documents are ranked higher when they have a larger number of elements in common with the query (see Chapter 2).

4.2.4 Natural Language

Pushing the fuzzy Boolean model even further, the distinction between AND and OR can be completely blurred, so that a query becomes simply an enumeration of words and context queries. All the documents matching a portion of the user query are retrieved. Higher ranking is assigned to those documents matching more parts of the query. The negation can be handled by letting the user express

† Notice that the same problem arises in the relational calculus, which is shown similar to the relational algebra only when 'unsafe' expressions are avoided. Unsafe expressions are those that make direct or indirect reference to a universe of elements, as NOT does.

that some words are not desired, so that the documents containing them are penalized in the ranking computation. A threshold may be selected so that the documents with very low weights are not retrieved. Under this scheme we have completely eliminated any reference to Boolean operations and entered into the field of natural language queries. In fact, one can consider that Boolean queries are a simplified abstraction of natural language queries.

A number of new issues arise once this model is used, especially those related to the proper way to rank an element with respect to a query. The search criterion can be re-expressed using a different model, where documents and queries are considered just as a vector of 'term weights' (with one coordinate per interesting keyword or even per existing text word) and queries are considered in exactly the same way (context queries are not considered in this case). Therefore, the query is now internally converted into a vector of term weights and the aim is to retrieve all the vectors (documents) which are *close* to the query (where closeness has to be defined in the model). This allows many interesting possibilities, for instance a complete document can be used as a query (since it is also a vector), which naturally leads to the use of relevance feedback techniques (i.e., the user can select a document from the result and submit it as a new query to retrieve documents similar to the selected one). The algorithms for this model are totally different from those based on searching patterns (it is even possible that not every text word needs to be searched but only a small set of hopefully representative keywords extracted from each document). Natural language querying is also covered in Chapter 14.

4.3 Pattern Matching

In this section we discuss more specific query formulations (based on the concept of a *pattern*) which allow the retrieval of pieces of text that have some property. These data retrieval queries are useful for linguistics, text statistics, and data extraction. Their result can be fed into the composition mechanism described above to form phrases and proximity queries, comprising what we have called *basic queries*. Basic queries can be combined using Boolean expressions. In this sense we can view these data retrieval capabilities as enhanced tools for information retrieval. However, it is more difficult to rank the result of a pattern matching expression.

A *pattern* is a set of syntactic features that must occur in a text segment. Those segments satisfying the pattern specifications are said to 'match' the pattern. We are interested in documents containing segments which match a given search pattern. Each system allows the specification of some types of patterns, which range from very simple (for example, words) to rather complex (such as regular expressions). In general, as more powerful is the set of patterns allowed, more involved are the queries that the user can formulate and more complex is the implementation of the search. The most used types of patterns are:

- **Words** A string (sequence of characters) which must be a word in the text (see section 4.2). This is the most basic pattern.
- **Prefixes** A string which must form the beginning of a text word. For instance, given the prefix '`comput`' all the documents containing words such as '`computer`,' '`computation`,' '`computing`,' etc. are retrieved.
- **Suffixes** A string which must form the termination of a text word. For instance, given the suffix '`ters`' all the documents containing words such as '`computers`,' '`testers`,' '`painters`,' etc. are retrieved.
- **Substrings** A string which can appear within a text word. For instance, given the substring '`tal`' all the documents containing words such as '`coastal`,' '`talk`,' '`metallic`,' etc. are retrieved. This query can be restricted to find the substrings inside words, or it can go further and search the substring anywhere in the text (in this case the query is not restricted to be a sequence of letters but can contain word separators). For instance, a search for '`any flow`' will match in the phrase '`...many flowers...`'
- **Ranges** A pair of strings which matches any word lying between them in lexicographical order. Alphabets are normally sorted, and this induces an order into the strings which is called *lexicographical order* (this is indeed the order in which words in a dictionary are listed). For instance, the range between words '`held`' and '`hold`' will retrieve strings such as '`hoax`' and '`hissing`.'
- **Allowing errors** A word together with an error threshold. This search pattern retrieves all text words which are 'similar' to the given word. The concept of similarity can be defined in many ways. The general concept is that the pattern or the text may have errors (coming from typing, spelling, or from optical character recognition software, among others), and the query should try to retrieve the given word and what are likely to be its erroneous variants. Although there are many models for similarity among words, the most generally accepted in text retrieval is the *Levenshtein distance*, or simply *edit distance*. The edit distance between two strings is the minimum number of character insertions, deletions, and replacements needed to make them equal (see Chapter 6). Therefore, the query specifies the maximum number of allowed errors for a word to match the pattern (i.e., the maximum allowed edit distance). This model can also be extended to search substrings (not only words), retrieving any text segment which is at the allowed edit distance from the search pattern. Under this extended model, if a typing error splits '`flower`' into '`flo wer`' it could still be found with one error, while in the restricted case of words it could not (since neither '`flo`' nor '`wer`' are at edit distance 1 from '`flower`'). Variations on this distance model are of use in computational biology for searching on DNA or protein sequences as well as in signal processing.
- **Regular expressions** Some text retrieval systems allow searching for *regular expressions*. A regular expression is a rather general pattern built

up by simple strings (which are meant to be matched as substrings) and the following operators:

- union: if e_1 and e_2 are regular expressions, then $(e_1|e_2)$ matches what e_1 or e_2 matches.
- concatenation: if e_1 and e_2 are regular expressions, the occurrences of $(e_1\ e_2)$ are formed by the occurrences of e_1 immediately followed by those of e_2 (therefore simple strings can be thought of as a concatenation of their individual letters).
- repetition: if e is a regular expression, then (e^*) matches a sequence of zero or more contiguous occurrences of e.

For instance, consider a query like '`pro (blem | tein) (s | ε) (0 | 1 | 2)*`' (where ε denotes the empty string). It will match words such as '`problem02`' and '`proteins.`' As in previous cases, the matches can be restricted to comprise a whole word, to occur inside a word, or to match an arbitrary text segment. This can also be combined with the previous type of patterns to search a regular expression allowing errors.

- **Extended patterns** It is normal to use a more user-friendly query language to represent some common cases of regular expressions. Extended patterns are subsets of the regular expressions which are expressed with a simpler syntax. The retrieval system can internally convert extended patterns into regular expressions, or search them with specific algorithms. Each system supports its own set of extended patterns, and therefore no formal definition exists. Some examples found in many new systems are:
  - classes of characters, i.e. one or more positions within the pattern are matched by any character from a pre-defined set. This involves features such as case-insensitive matching, use of ranges of characters (e.g., specifying that some character must be a digit), complements (e.g., some character must not be a letter), enumeration (e.g., a character must be a vowel), wild cards (i.e., a position within the pattern matches with anything), among others.
  - conditional expressions, i.e., a part of the pattern may or may not appear.
  - wild characters which match any sequence in the text, e.g. any word which starts as '`flo`' and ends with '`ers`,' which matches '`flowers`' as well as '`flounders`.'
  - combinations that allow some parts of the pattern to match exactly and other parts with errors.

4.4 Structural Queries

Up to now we have considered the text collection as a set of documents which can be queried with regard to their text content. This model is unable to take advantage of novel text features which are becoming commonplace, such as the

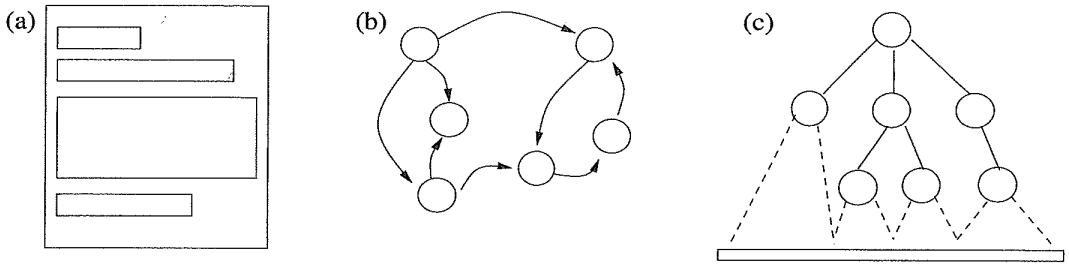


Figure 4.2 The three main structures: (*a*) form-like fixed structure, (*b*) hypertext structure, and (*c*) hierarchical structure.

text structure. The text collections tend to have some structure built into them, and allowing the user to query those texts based on their structure (and not only their content) is becoming attractive. The standardization of languages to represent structured texts such as HTML has pushed forward in this direction (see Chapter 6).

As discussed in Chapter 2, mixing contents and structure in queries allows us to pose very powerful queries, which are much more expressive than each query mechanism by itself. By using a query language that integrates both types of queries, the retrieval quality of textual databases can be improved.

This mechanism is built on top of the basic queries, so that they select a set of documents that satisfy certain constraints on their content (expressed using words, phrases, or patterns that the documents must contain). On top of this, some structural constraints can be expressed using containment, proximity, or other restrictions on the structural elements (e.g., chapters, sections, etc.) present in the documents. The Boolean queries can be built on top of the structural queries, so that they combine the sets of documents delivered by those queries. In the Boolean syntax tree (recall the example of Figure 4.1) the structural queries form the leaves of the tree. On the other hand, structural queries can themselves have a complex syntax.

We divide this section according to the type of structures found in text databases. Figure 4.2 illustrates them. Although structured query languages should be amenable for ranking, this is still an open problem.

In what follows it is important to distinguish the difference between the structure that a text may *have* and what can be *queried* about that structure. In general, natural language texts may have any desired structure. However, different models allow the querying of only some aspects of the real structure. When we say that the structure allowed is restricted in some way, we mean that only the aspects which follow this restriction can be queried, albeit the text may have more structural information. For instance, it is possible that an article has a nested structure of sections and subsections, but the query model does not accept recursive structures. In this case we will not be able to query for sections included in others, although this may be the case in the texts documents under consideration.

4.4.1 Fixed Structure

The structure allowed in texts was traditionally quite restrictive. The documents had a fixed set of *fields*, much like a filled form. Each field had some text inside. Some fields were not present in all documents. Only rarely could the fields appear in any order or repeat across a document. A document could not have text not classified under any field. Fields were not allowed to nest or overlap. The retrieval activity allowed on them was restricted to specifying that a given basic pattern was to be found only in a given field. Most current commercial systems use this model.

This model is reasonable when the text collection has a fixed structure. For instance, a mail archive could be regarded as a set of mails, where each mail has a sender, a receiver, a date, a subject, and a body field. The user can thus search for the mails sent to a given person with '`footbal1`' in the subject field. However, the model is inadequate to represent the hierarchical structure present in an HTML document, for instance.

If the division of the text into fields is rigid enough, the content of some fields can even be interpreted not as text but as numbers, dates, etc. thereby allowing different queries to be posed on them (e.g., month ranges in dates). It is not hard to see that this idea leads naturally to the relational model, each field corresponding to a column in the database table. Looking at the database as a text allows us to query the textual fields with much more power than is common in relational database systems. On the other hand, relational databases may make better use of their knowledge on the data types involved to build specialized and more efficient indices. A number of approaches towards combining these trends have been proposed in recent years, their main problem being that they do not achieve optimal performance because the text is usually stored together with other types of data. Nevertheless, there are several proposals that extend SQL (Structured Query Language) to allow full-text retrieval. Among them we can mention proposals by leading relational database vendors such as Oracle and Sybase, as well as SFQL, which is covered in section 4.5.

4.4.2 Hypertext

Hypertexts probably represent the maximum freedom with respect to structuring power. A hypertext is a directed graph where the nodes hold some text and the links represent connections between nodes or between positions inside the nodes (see Chapter 2). Hypertexts have received a lot of attention since the explosion of the Web, which is indeed a gigantic hypertext-like database spread across the world.

However, retrieval from a hypertext began as a merely navigational activity. That is, the user had to manually traverse the hypertext nodes following links to search what he wanted. It was not possible to query the hypertext based on its structure. Even in the Web one can search by the text contents of the nodes, but not by their structural connectivity.

An interesting proposal to combine browsing and searching on the Web is WebGlimpse. It allows classical navigation plus the ability to search by content in the neighborhood of the current node. Currently, some query tools have appeared that achieve the goal of querying hypertexts based on their content and their structure. This problem is covered in detail in Chapter 13.

4.4.3 Hierarchical Structure

An intermediate structuring model which lies between fixed structure and hypertext is the hierarchical structure. This model represents a recursive decomposition of the text and is a natural model for many text collections (e.g., books, articles, legal documents, structured programs, etc.). Figure 4.3 shows an example of such a hierarchical structure.

The simplification from hypertext to a hierarchy allows the adoption of faster algorithms to solve queries. As a general rule, the more powerful the model, the less efficiently it can be implemented.

Our aim in this section is to analyze and discuss the different approaches presented by the hierarchical models. We first present a selection of the most representative models and then discuss the main subjects of this area.

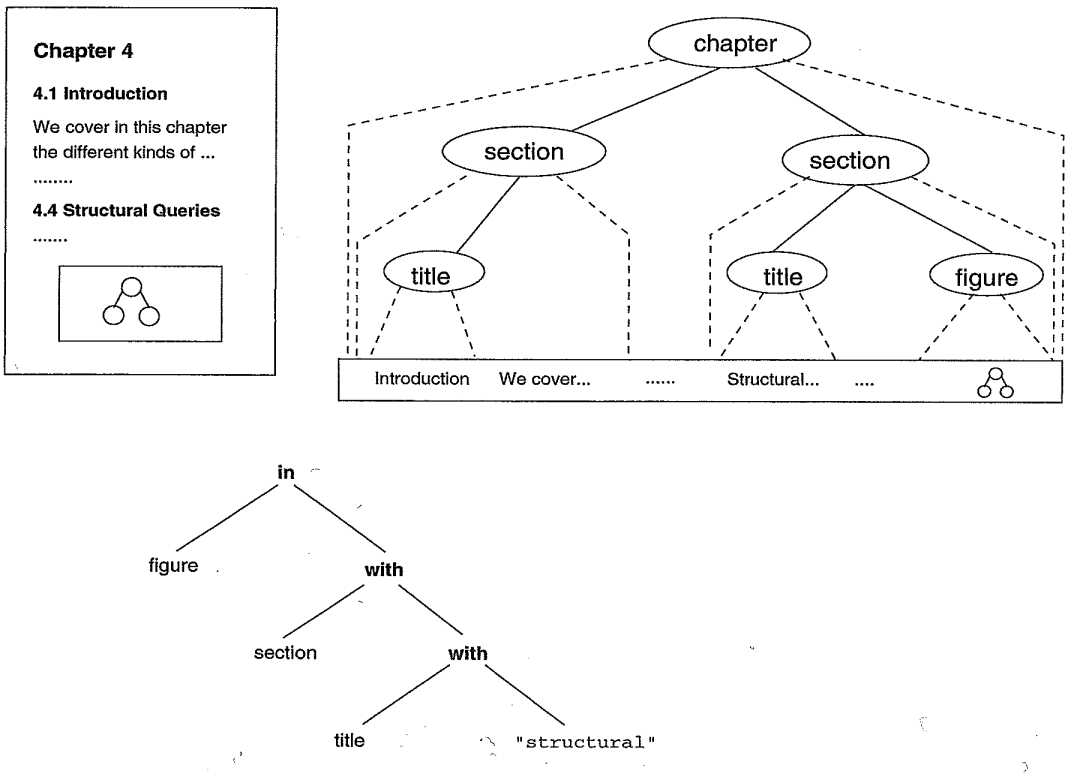


Figure 4.3 An example of a hierarchical structure: the page of a book, its schematic view, and a parsed query to retrieve the figure.

A Sample of Hierarchical Models

PAT Expressions
These are built on the same index as the text index, i.e. there is no special separate index on the structure. The structure is assumed to be marked in the text by *tags* (as in HTML), and therefore is defined in terms of initial and final tags. This allows a dynamic scheme where the structure of interest is not fixed but can be determined at query time. For instance, since tags need not to be especially designed as normal tags, one can define that the end-of-lines are the marks in order to define a structure on lines. This also allows for a very efficient implementation and no additional space overhead for the structure.

Each pair of initial and final tags defines a *region*, which is a set of contiguous text areas. Externally computed regions are also supported. However, the areas of a region cannot nest or overlap, which is quite restrictive. There is no restriction on areas of different regions.

Apart from text searching operations, it is possible to select areas containing (or not) other areas, contained (or not) in other areas, or followed (or not) by other areas.

A disadvantage is that the algebra mixes regions and sets of text positions which are incompatible and force complex conversion semantics. For instance, if the result of a query is going to generate overlapping areas (a fact that cannot be determined beforehand) then the result is converted to positions. Also, the dynamic definition of regions is flexible but requires the structure to be expressable using tags (also called 'markup', see Chapter 6), which for instance does not occur in some structured programming languages.

Overlapped Lists
These can be seen as an evolution of PAT Expressions. The model allows for the areas of a region to overlap, but not to nest. This elegantly solves the problems of mixing regions and sets of positions. The model considers the use of an inverted list (see Chapter 8) where not only the words but also the regions are indexed.

Apart from the operations of PAT Expressions, the model allows us to perform set union, and to combine regions. Combination means selecting the minimal text areas which include any two areas taken from two regions. A 'followed by' operator imposes the additional restriction that the first area must be before the second one. An 'n words' operator generates the region of all (overlapping) sequences of n words of the text (this is further used to retrieve elements close to each other). If an operation produces a region with nested areas, only the minimal areas are selected. An example is shown in Figure 2.11.

The implementation of this model can also be very efficient. It is not clear, however, whether overlapping is good or not for capturing the structural properties that information has in practice. A new proposal allows the structure to be nested *and* overlapped, showing that more interesting operators can still be implemented.

Lists of References

These are an attempt to make the definition and querying of structured text uniform, using a common language. The language goes beyond querying structured text, so we restrict our attention to the subset in which we are interested.

The structure of documents is fixed and hierarchical, which makes it impossible to have overlapping results. All possible regions are defined at indexing time. The answers delivered are more restrictive, since nesting is not allowed (only the top-level elements qualify) and all elements must be of the same type, e.g. only sections, or only paragraphs. In fact, there are also hypertext links but these cannot be queried (the model also has navigational features).

A static hierarchical structure makes it possible to speak in terms of direct ancestry of nodes, a concept difficult to express when the structure is dynamic. The language allows for querying on 'path expressions,' which describe paths in the structure tree.

Answers to queries are seen as lists of 'references.' A reference is a pointer to a region of the database. This integrates in an elegant way answers to queries and hypertext links, since all are lists of references.

Proximal Nodes

This model tries to find a good compromise between expressiveness and efficiency. It does not define a specific language, but a model in which it is shown that a number of useful operators can be included achieving good efficiency.

The structure is fixed and hierarchical. However, many independent structures can be defined on the same text, each one being a strict hierarchy but allowing overlaps between areas of different hierarchies. An example is shown in Figure 2.12.

A query can relate different hierarchies, but returns a subset of the nodes of one hierarchy only (i.e., nested elements are allowed in the answers, but no overlaps). Text matching queries are modeled as returning nodes from a special 'text hierarchy.'

The model specifies a fully compositional language where the leaves of the query syntax tree are formed by basic queries on contents or names of structural elements (e.g., all chapters). The internal nodes combine results. For efficiency, the operations defined at the internal nodes must be implementable looking at the identity and text areas of the operands, and must relate nodes which are close in the text.

It has been shown that many useful operators satisfy this restriction: selecting elements that (directly or transitively) include or are included in others; that are included at a given position (e.g., the third paragraph of each chapter); that are shortly before or after others; set manipulation; and many powerful variations. Operations on content elements deliver a set of regions with no nesting, and those results can be fully integrated into any query. This ability to integrate the text into the model is very useful. On the other hand, some queries requiring non-proximal operations are not allowed, for instance *semijoins*. An example of a semijoin is 'give me the titles of all the chapters referenced in this chapter.'

Tree Matching
This model relies on a single primitive: tree inclusion, whose main idea is as follows. Interpreting the structure both of the text database and of the query (which is defined as a pattern on the structure) as trees, determine an embedding of the query into the database which respects the hierarchical relationships between nodes of the query.

Two variants are studied. *Ordered inclusion* forces the embedding to respect the left-to-right relations among siblings in the query, while *unordered inclusion* does not. The leaves of the query can be not only structural elements but also text patterns, meaning that the ancestor of the leaf must contain that pattern.

Simple queries return the roots of the matches. The language is enriched by Prolog-like variables, which can be used to express requirements on equality between parts of the matched substructure and to retrieve another part of the match, not only the root. Logical variables are also used for union and intersection of queries, as well as to emulate tuples and join capabilities.

Although the language is set oriented, the algorithms work by sequentially obtaining each match. The use of logical variables and unordered inclusion makes the search problem intractable (NP-hard in many cases). Even the good cases have an inefficient solution in practice.

Discussion

A survey of the main hierarchical models raises a number of interesting issues, most of them largely unresolved up to now. Some of them are listed below.

Static or dynamic structure
As seen, in a static structure there are one or more explicit hierarchies (which can be queried, e.g., by ancestry), while in a dynamic structure there is not really a hierarchy, but the required elements are built on the fly. A dynamic structure is implemented over a normal text index, while a static one may or may not be. A static structure is independent of the text markup, while a dynamic one is more flexible for building arbitrary structures.

Restrictions on the structure
The text or the answers may have restrictions about nesting and/or overlapping. In some cases these restrictions exist for efficiency reasons. In other cases, the query language is restricted to avoid restricting the structure. This choice is largely dependent on the needs of each application.

Integration with text
In many structured models, the text content is merely seen as a secondary source of information which is used only to restrict the matches of structural elements. In classic IR models, on the other side, information on the structure is the secondary element which is used only to restrict text matches. For an effective

integration of queries on text content with queries on text structure, the query language must provide for full expressiveness of both types of queries and for effective means of combining them.

Query language
Typical queries on structure allow the selection of areas that contain (or not) other areas, that are contained (or not) in other areas, that follow (or are followed by) other areas, that are close to other areas, and set manipulation. Many of them are implemented in most models, although each model has unique features. Some kind of standardization, expressiveness taxonomy, or formal categorization would be highly desirable but does not exist yet.

4.5 Query Protocols

In this section we briefly cover some query languages that are used automatically by software applications to query text databases. Some of them are proposed as standards for querying CD-ROMs or as intermediate languages to query library systems. Because they are not intended for human use, we refer to them as protocols rather than languages. More information on protocols can be found in Chapters 14 and 15. The most important query protocols are:

- **Z39.50** is a protocol approved as a standard in 1995 by ANSI and NISO. This protocol is intended to query bibliographical information using a standard interface between the client and the host database manager which is independent of the client user interface and of the query database language at the host. The database is assumed to be a text collection with some fixed fields (although it is more flexible than usual). The Z39.50 protocol is used broadly and is part, for instance, of WAIS (see below). The protocol does not only specify the query language and its semantics, but also the way in which client and server establish a session, communicate and exchange information, etc. Although originally conceived only to operate on bibliographical information (using the Machine Readable Cataloging Record (MARC) format), it has been extended to query other types of information as well.
- **WAIS** (Wide Area Information Service) is a suite of protocols that was popular at the beginning of the 1990s before the boom of the Web. The goal of WAIS was to be a network publishing protocol and to be able to query databases through the Internet.

In the CD-ROM publishing arena, there are several proposals for query protocols. The main goal of these protocols is to provide 'disk interchangeability.' This means more flexibility in data communication between primary information providers and end users. It also enables significant cost savings since it allows access to diverse information without the need to buy, install, and train users for different data retrieval applications. We briefly cover three of these proposals:

- **CCL** (Common Command Language) is a NISO proposal (Z39.58 or ISO 8777) based on Z39.50. It defines 19 commands that can be used interactively. It is more popular in Europe, although very few products use it. It is based on the classical Boolean model.
- **CD-RDx** (Compact Disk Read only Data exchange) uses a client-server architecture and has been implemented in most platforms. The client is generic while the server is designed and provided by the CD-ROM publisher who includes it with the database in the CD-ROM. It allows fixed-length fields, images, and audio, and is supported by such US national agencies as the CIA, NASA, and GSA.
- **SFQL** (Structured Full-text Query Language) is based on SQL and also has a client-server architecture. SFQL has been adopted as a standard by the aerospace community (the Air Transport Association/Aircraft Industry Association). Documents are rows in a relational table and can be tagged using SGML. The language defines the format of the answer, which has a header and a variable length message area. The language does not define any specific formatting or markup. For example, a query in SFQL is:

  ```
  Select abstract from journal.papers where title
  contains "text search"
  ```

  The language supports Boolean and logical operators, thesaurus, proximity operations, and some special characters such as wild cards and repetition. For example:

  ```
  ...  where paper contains "retrieval" or like "info
  %" and date > 1/1/98
  ```

  Compared with CCL or CD-RDx, SFQL is more general and flexible, although it is based on a relational model, which is not always the best choice for a document database.

4.6 Trends and Research Issues

We reviewed in this chapter the main aspects of the query languages that retrieve information from textual databases. Our discussion covered from the most classic tools to the most novel capabilities that are emerging, from searching words to extended patterns, from the Boolean model to querying structures. Table 4.1 shows the different basic queries allowed in the different models. Although the probabilistic and the Bayesian belief network (BBN) models are based on word queries, they can incorporate set operations.

We present in Figure 4.4 the types of operations we covered and how they can be structured (not all of them exist in all models and not all of them have to be used to form a query). The figure shows, for instance, that we can form a query using Boolean operations over phrases (skipping structural queries), which

Model	Queries allowed
Boolean	word, set operations
Vector	words
Probabilistic	words
BBN	words

Table 4.1 Relationship between types of queries and models.

can be formed by words and by regular expressions (skipping the ability to allow errors).

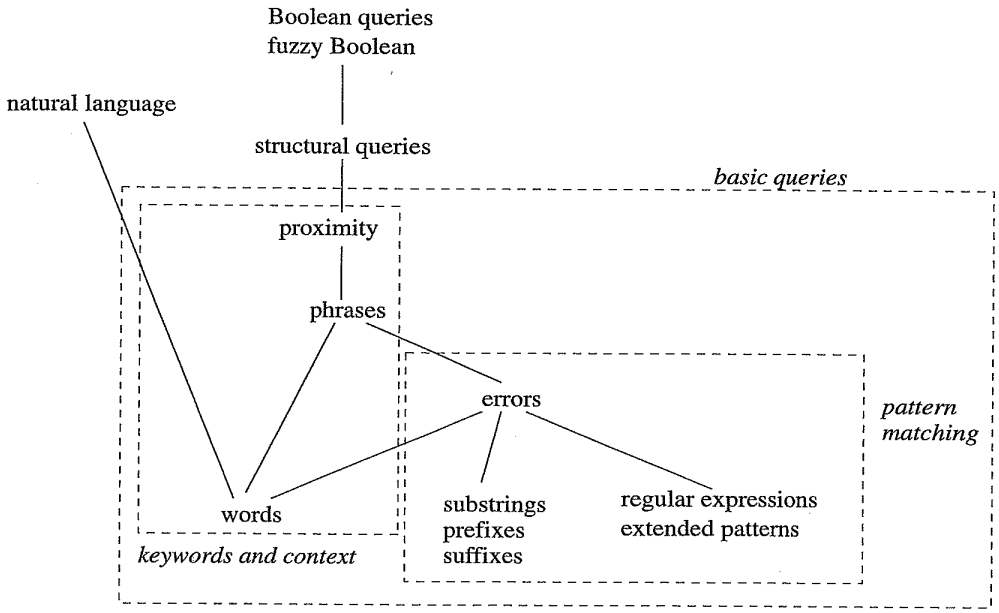


Figure 4.4 The types of queries covered and how they are structured.

The area of query languages for text databases is definitely moving towards higher flexibility. While text models are moving towards the goal of achieving a better understanding of the user needs (by providing relevance feedback, for instance), the query languages are allowing more and more power in the specification of the query. While extended patterns and searching allowing errors permit us to find patterns without complete knowledge of what is wanted, querying on the structure of the text (and not only on its content) provides greater expressiveness and increased functionality.

Another important research topic is visual query languages. Visual metaphors can help non-experienced users to pose complex Boolean queries. Also, a visual query language can include the structure of the document. This topic is related to user interfaces and visualization and is covered in Chapter 10.

4.7 Bibliographic Discussion

The material on classical query languages (most simple patterns, Boolean model, and fixed structure) is based on current commercial systems, such as Fulcrum, Verity, and others, as well as on non-commercial systems such as Glimpse [540] and Igrep [26].

The fuzzy Boolean model is described in [703]. The Levenshtein distance is described in [504] and [25]. Soundex is explained in [445]. A comparison of the effectiveness of different similarity models is given in [595]. A good source on regular expressions is [375]. A rich language on extended patterns is described in [837].

A classical reference on hypertext is [181]. The WebGlimpse system is presented in [539]. The discussion of hierarchical text is partially based on [41]. The original proposals are: PAT Expressions [693], Overlapped Lists [173] and the new improved proposal [206], Lists of References [534], Proximal Nodes [590], and Tree Matching [439]. PAT Expressions are the basic model of the PAT Text Searching System [309]. A simple structured text model is presented in [36] and a visual query language that includes structure is discussed in [44].

More information on Z39.50 can be obtained from [23]. More information on WAIS is given in [425]. For details on SFQL see [392].

Chapter 5
Query Operations

5.1 Introduction

Without detailed knowledge of the collection make-up and of the retrieval environment, most users find it difficult to formulate queries which are well designed for retrieval purposes. In fact, as observed with Web search engines, the users might need to spend large amounts of time reformulating their queries to accomplish effective retrieval. This difficulty suggests that the first query formulation should be treated as an initial (naive) attempt to retrieve relevant information. Following that, the documents initially retrieved could be examined for relevance and new improved query formulations could then be constructed in the hope of retrieving additional useful documents. Such query reformulation involves two basic steps: expanding the original query with new terms and reweighting the terms in the expanded query.

In this chapter, we examine a variety of approaches for improving the initial query formulation through query expansion and term reweighting. These approaches are grouped in three categories: (a) approaches based on feedback information from the user; (b) approaches based on information derived from the set of documents initially retrieved (called the *local* set of documents); and (c) approaches based on global information derived from the document collection. In the first category, user relevance feedback methods for the vector and probabilistic models are discussed. In the second category, two approaches for local analysis (i.e., analysis based on the set of documents initially retrieved) are presented. In the third category, two approaches for global analysis are covered.

Our discussion is not aimed at completely covering the area, neither does it intend to present an exhaustive survey of query operations. Instead, our discussion is based on a selected bibliography which, we believe, is broad enough to allow an overview of the main issues and tradeoffs involved in query operations. Local and global analysis are highly dependent on clustering algorithms. Thus, clustering is covered throughout our discussion. However, there is no intention of providing a complete survey of clustering algorithms for information retrieval.

5.2 User Relevance Feedback

Relevance feedback is the most popular query reformulation strategy. In a relevance feedback cycle, the user is presented with a list of the retrieved documents and, after examining them, marks those which are relevant. In practice, only the top 10 (or 20) ranked documents need to be examined. The main idea consists of selecting important terms, or expressions, attached to the documents that have been identified as relevant by the user, and of enhancing the importance of these terms in a new query formulation. The expected effect is that the new query will be moved towards the relevant documents and away from the non-relevant ones.

Early experiments using the Smart system [695] and later experiments using the probabilistic weighting model [677] have shown good improvements in precision for small test collections when relevance feedback is used. Such improvements come from the use of two basic techniques: query expansion (addition of new terms from relevant documents) and term reweighting (modification of term weights based on the user relevance judgement).

Relevance feedback presents the following main advantages over other query reformulation strategies: (a) it shields the user from the details of the query reformulation process because all the user has to provide is a relevance judgement on documents; (b) it breaks down the whole searching task into a sequence of small steps which are easier to grasp; and (c) it provides a controlled process designed to emphasize some terms (relevant ones) and de-emphasize others (non-relevant ones).

In the following three subsections, we discuss the usage of user relevance feedback to (a) expand queries with the vector model, (b) reweight query terms with the probabilistic model, and (c) reweight query terms with a variant of the probabilistic model.

5.2.1 Query Expansion and Term Reweighting for the Vector Model

The application of *relevance feedback* to the vector model considers that the term-weight vectors of the documents identified as relevant (to a given query) have similarities among themselves (i.e., relevant documents resemble each other). Further, it is assumed that non-relevant documents have term-weight vectors which are dissimilar from the ones for the relevant documents. The basic idea is to reformulate the query such that it gets closer to the term-weight vector space of the relevant documents.

Let us define some additional terminology regarding the processing of a given query q as follows,

D_r: set of relevant documents, as identified by the user, among the *retrieved* documents;
D_n: set of non-relevant documents among the *retrieved* documents;
C_r: set of relevant documents among all documents in the collection;

$|D_r|, |D_n|, |C_r|$: number of documents in the sets D_r, D_n, and C_r, respectively;
α, β, γ: tuning constants.

Consider first the unrealistic situation in which the complete set C_r of relevant documents to a given query q is known in advance. In such a situation, it can be demonstrated that the best query vector for distinguishing the relevant documents from the non-relevant documents is given by,

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\forall \vec{d}_j \in C_r} \vec{d}_j - \frac{1}{N - |C_r|} \sum_{\forall \vec{d}_j \notin C_r} \vec{d}_j \qquad \textbf{(5.1)}$$

The problem with this formulation is that the *relevant* documents which compose the set C_r are not known a priori. In fact, we are looking for them. The natural way to avoid this problem is to formulate an initial query and to incrementally change the initial query vector. This incremental change is accomplished by restricting the computation to the documents *known* to be relevant (according to the user judgement) at that point. There are three classic and similar ways to calculate the modified query $\vec{q}_m$ as follows,

$$\begin{aligned}
Standard_Rochio: \quad \vec{q}_m &= \alpha\, \vec{q} + \frac{\beta}{|D_r|} \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \frac{\gamma}{|D_n|} \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j \\
Ide_Regular: \quad \vec{q}_m &= \alpha\, \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma \sum_{\forall \vec{d}_j \in D_n} \vec{d}_j \\
Ide_Dec_Hi: \quad \vec{q}_m &= \alpha\, \vec{q} + \beta \sum_{\forall \vec{d}_j \in D_r} \vec{d}_j - \gamma\ max_{non\text{-}relevant}(\vec{d}_j)
\end{aligned}$$

where $max_{non-relevant}(\vec{d}_j)$ is a reference to the highest ranked non-relevant document. Notice that now D_r and D_n stand for the sets of relevant and non-relevant documents (among the retrieved ones) according to the user judgement, respectively. In the original formulations, Rochio [678] fixed $\alpha = 1$ and Ide [391] fixed $\alpha = \beta = \gamma = 1$. The expressions above are modern variants. The current understanding is that the three techniques yield similar results (in the past, Ide Dec-Hi was considered slightly better).

The Rochio formulation is basically a direct adaptation of equation 5.1 in which the terms of the original query are added in. The motivation is that in practice the original query q may contain important information. Usually, the information contained in the relevant documents is more important than the information provided by the non-relevant documents [698]. This suggests making the constant γ smaller than the constant β. An alternative approach is to set γ to 0 which yields a *positive* feedback strategy.

The main advantages of the above relevance feedback techniques are simplicity and good results. The simplicity is due to the fact that the modified term weights are computed directly from the set of retrieved documents. The good

results are observed experimentally and are due to the fact that the modified query vector does reflect a portion of the intended query semantics. The main disadvantage is that *no* optimality criterion is adopted.

5.2.2 Term Reweighting for the Probabilistic Model

The probabilistic model dynamically ranks documents similar to a query q according to the probabilistic ranking principle. From Chapter 2, we already know that the similarity of a document d_j to a query q can be expressed as

$$sim(d_j, q) \; \alpha \; \sum_{i=1}^{t} w_{i,q} \; w_{i,j} \; \left(\log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\overline{R})}{P(k_i|\overline{R})} \right) \qquad \textbf{(5.2)}$$

where $P(k_i|R)$ stands for the probability of observing the term k_i in the set R of relevant documents and $P(k_i|\overline{R})$ stands for the probability of observing the term k_i in the set $\overline{R}$ of non-relevant documents.

Initially, equation 5.2 cannot be used because the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$ are unknown. A number of different methods for estimating these probabilities automatically (i.e., without feedback from the user) were discussed in Chapter 2. With user feedback information, these probabilities are estimated in a slightly different way as follows.

For the initial search (when there are no retrieved documents yet), assumptions often made include: (a) $P(k_i|R)$ is constant for all terms k_i (typically 0.5) and (b) the term probability distribution $P(k_i|\overline{R})$ can be approximated by the distribution in the whole collection. These two assumptions yield:

$$\begin{aligned} P(k_i|R) &= 0.5 \\ P(k_i|\overline{R}) &= \frac{n_i}{N} \end{aligned}$$

where, as before, n_i stands for the number of documents in the collection which contain the term k_i. Substituting into equation 5.2, we obtain

$$sim_{initial}(d_j, q) = \sum_{i}^{t} w_{i,q} \; w_{i,j} \; \log \frac{N - n_i}{n_i}$$

For the feedback searches, the accumulated statistics related to the relevance or non-relevance of previously retrieved documents are used to evaluate the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$. As before, let D_r be the set of relevant retrieved documents (according to the user judgement) and $D_{r,i}$ be the subset of D_r composed of the documents which contain the term k_i. Then, the probabilities $P(k_i|R)$ and $P(k_i|\overline{R})$ can be approximated by

$$P(k_i|R) = \frac{|D_{r,i}|}{|D_r|}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}|}{N - |D_r|} \qquad \textbf{(5.3)}$$

Using these approximations, equation 5.2 can rewritten as

$$sim(d_j, q) = \sum_{i=1}^{t} w_{i,q}\ w_{i,j}\ \log\left[\frac{|D_{r,i}|}{|D_r| - |D_{r,i}|} \div \frac{n_i - |D_{r,i}|}{N - |D_r| - (n_i - |D_{r,i}|)}\right]$$

Notice that here, contrary to the procedure in the vector space model, no query expansion occurs. The same query terms are being reweighted using feedback information provided by the user.

Formula 5.3 poses problems for certain small values of $|D_r|$ and $|D_{r,i}|$ that frequently arise in practice ($|D_r| = 1, |D_{r,i}| = 0$). For this reason, a 0.5 adjustment factor is often added to the estimation of $P(k_i|R)$ and $P(k_i|\overline{R})$ yielding

$$P(k_i|R) = \frac{|D_{r,i}| + 0.5}{|D_r| + 1}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}| + 0.5}{N - |D_r| + 1} \tag{5.4}$$

This 0.5 adjustment factor may provide unsatisfactory estimates in some cases, and alternative adjustments have been proposed such as n_i/N or $(n_i - |D_{r,i}|)$ $/(N - |D_r|)$ [843]. Taking n_i/N as the adjustment factor (instead of 0.5), equation 5.4 becomes

$$P(k_i|R) = \frac{|D_{r,i}| + \frac{n_i}{N}}{|D_r| + 1}; \quad P(k_i|\overline{R}) = \frac{n_i - |D_{r,i}| + \frac{n_i}{N}}{N - |D_r| + 1}$$

The main advantages of this relevance feedback procedure are that the feedback process is directly related to the derivation of new weights for *query* terms and that the term reweighting is optimal under the assumptions of term independence and binary document indexing ($w_{i,q} \in \{0, 1\}$ and $w_{i,j} \in \{0, 1\}$). The disadvantages include: (1) document term weights are *not* taken into account during the feedback loop; (2) weights of terms in the previous query formulations are also disregarded; and (3) no query expansion is used (the same set of index terms in the original query is reweighted over and over again). As a result of these disadvantages, the probabilistic relevance feedback methods do *not* in general operate as effectively as the conventional vector modification methods.

To extend the probabilistic model with query expansion capabilities, different approaches have been proposed in the literature ranging from term weighting for query expansion to term clustering techniques based on spanning trees. All of these approaches treat probabilistic query expansion separately from probabilistic term reweighting. While we do not discuss them here, a brief history of research on this issue and bibliographical references can be found in section 5.6.

5.2.3 A Variant of Probabilistic Term Reweighting

The discussion above on term reweighting is based on the classic probabilistic model introduced by Robertson and Sparck Jones in 1976. In 1983, Croft extended this weighting scheme by suggesting distinct initial search methods

and by adapting the probabilistic formula to include within-document frequency weights. This variant of probabilistic term reweighting is more flexible (and also more powerful) and is briefly reviewed in this section.

The formula 5.2 for probabilistic ranking can be rewritten as

$$sim(d_j, q) \; \alpha \; \sum_{i=1}^{t} w_{i,q} \; w_{i,j} \; F_{i,j,q}$$

where $F_{i,j,q}$ is interpreted as a factor which depends on the triple $[k_i, d_j, q]$. In the classic formulation, $F_{i,j,q}$ is computed as a function of $P(k_i|R)$ and $P(k_i|\overline{R})$ (see equation 5.2). In his variant, Croft proposed that the initial search and the feedback searches use distinct formulations.

For the initial search, he suggested

$$\begin{aligned} F_{i,j,q} &= (C + idf_i) \; \overline{f}_{i,j} \\ \overline{f}_{i,j} &= K + (1 + K) \; \frac{f_{i,j}}{max(f_{i,j})} \end{aligned}$$

where $\overline{f}_{i,j}$ is a normalized within-document frequency. The parameters C and K should be adjusted according to the collection. For automatically indexed collections, C should be initially set to 0.

For the feedback searches, Croft suggested the following formulation for $F_{i,j,q}$

$$F_{i,j,q} = \left(C + \log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\overline{R})}{P(k_i|\overline{R})} \right) \; \overline{f}_{i,j}$$

where $P(k_i|R)$ and $P(k_i|\overline{R})$ are computed as in equation 5.4.

This variant of probabilistic term reweighting has the following advantages: (1) it takes into account the within-document frequencies; (2) it adopts a normalized version of these frequencies; and (3) it introduces the constants C and K which provide for greater flexibility. However, it constitutes a more complex formulation and, as before, it operates solely on the terms originally in the query (without query expansion).

5.2.4 Evaluation of Relevance Feedback Strategies

Consider the modified query vector $\vec{q}_m$ generated by the Rochio formula and assume that we want to evaluate its retrieval performance. A simplistic approach is to retrieve a set of documents using $\vec{q}_m$, to rank them using the vector formula, and to measure recall-precision figures relative to the set of relevant documents (provided by the experts) for the original query vector $\vec{q}$. In general, the results show spectacular improvements. Unfortunately, a significant part of this improvement results from the higher ranks assigned to the set R of documents

already identified as relevant during the feedback process [275]. Since the user has seen these documents already (and pointed them as relevants), such evaluation is unrealistic. Further, it masks any real gains in retrieval performance due to documents not seen by the user yet.

A more realistic approach is to evaluate the retrieval performance of the modified query vector $\vec{q}_m$ considering only the *residual collection* i.e., the set of all documents minus the set of feedback documents provided by the user. Because highly ranked documents are removed from the collection, the recall-precision figures for $\vec{q}_m$ tend to be lower than the figures for the original query vector $\vec{q}$. This is not a limitation because our main purpose is to compare the performance of distinct relevance feedback strategies (and not to compare the performance before and after feedback). Thus, as a basic rule of thumb, any experimentation involving relevance feedback strategies should always evaluate recall-precision figures relative to the residual collection.

5.3 Automatic Local Analysis

In a user relevance feedback cycle, the user examines the top ranked documents and separates them into two classes: the relevant ones and the non-relevant ones. This information is then used to select new terms for query expansion. The reasoning is that the expanded query will retrieve more relevant documents. Thus, there is an underlying notion of *clustering* supporting the feedback strategy. According to this notion, known relevant documents contain terms which can be used to describe a larger cluster of relevant documents. In this case, the description of this larger cluster of relevant documents is built interactively with assistance from the user.

A distinct approach is to attempt to obtain a description for a larger cluster of relevant documents automatically. This usually involves identifying terms which are related to the query terms. Such terms might be synonyms, stemming variations, or terms which are close to the query terms in the text (i.e., terms with a distance of at most k words from a query term). Two basic types of strategies can be attempted: global ones and local ones.

In a global strategy, all documents in the collection are used to determine a global thesaurus-like structure which defines term relationships. This structure is then shown to the user who selects terms for query expansion. Global strategies are discussed in section 5.4.

In a local strategy, the documents retrieved for a given query q are examined at query time to determine terms for query expansion. This is similar to a relevance feedback cycle but might be done without assistance from the user (i.e., the approach might be fully automatic). Two local strategies are discussed below: local clustering and local context analysis. The first is based on the work done by Attar and Fraenkel in 1977 and is used here to establish many of the fundamental ideas and concepts regarding the usage of clustering for query expansion. The second is a recent work done by Xu and Croft in 1996 and illustrates the advantages of combining techniques from both local and global analysis.

5.3.1 Query Expansion Through Local Clustering

Adoption of clustering techniques for query expansion is a basic approach which has been attempted since the early years of information retrieval. The standard approach is to build global structures such as association matrices which quantify term correlations (for instance, number of documents in which two given terms co-occur) and to use correlated terms for query expansion. The main problem with this strategy is that there is not consistent evidence that global structures can be used effectively to improve retrieval performance with general collections. One main reason seems to be that global structures do not adapt well to the local context defined by the current query. One approach to deal with this effect is to devise strategies which aim at optimizing the current search. Such strategies are based on *local clustering* and are now discussed. Our discussion is based on the original work by Attar and Fraenkel which appeared in 1977.

We first define basic terminology as follows.

Definition *Let $V(s)$ be a non-empty subset of words which are grammatical variants of each other. A canonical form s of $V(s)$ is called a* stem. *For instance, if $V(s)$=$\{$polish,polishing,polished$\}$ then s=polish.*

For a detailed discussion on stemming algorithms see Chapter 7. While stems are adopted in our discussion, the ideas below are also valid for non-stemmed keywords. We proceed with a characterization of the local nature of the strategies covered here.

Definition *For a given query q, the set D_l of documents retrieved is called the local document set. Further, the set V_l of all distinct words in the local document set is called the local vocabulary. The set of all distinct stems derived from the set V_l is referred to as S_l.*

We operate solely on the documents retrieved for the current query. Since it is frequently necessary to access the text of such documents, the application of local strategies to the Web is unlikely at this time. In fact, at a client machine, retrieving the text of 100 Web documents for local analysis would take too long, reducing drastically the interactive nature of Web interfaces and the satisfaction of the users. Further, at the search engine site, analyzing the text of 100 Web documents would represent an extra spending of CPU time which is not cost effective at this time (because search engines depend on processing a high number of queries per unit of time for economic survival). However, local strategies might be quite useful in the environment of intranets such as, for instance, the collection of documents issued by a large business company. Further, local strategies might also be of great assistance for searching information in specialized document collections (for instance, medical document collections).

Local feedback strategies are based on expanding the query with terms correlated to the query terms. Such correlated terms are those present in local clusters built from the local document set. Thus, before we discuss local

query expansion, we discuss strategies for building local clusters. Three types of clusters are covered: association clusters, metric clusters, and scalar clusters.

Association Clusters

An association cluster is based on the co-occurrence of stems (or terms) inside documents. The idea is that stems which co-occur frequently inside documents have a synonymity association. Association clusters are generated as follows.

Definition *The frequency of a stem s_i in a document d_j, $d_j \in D_l$, is referred to as $f_{s_i,j}$. Let $\vec{m}=(m_{ij})$ be an association matrix with $|S_l|$ rows and $|D_l|$ columns, where $m_{ij}=f_{s_i,j}$. Let $\vec{m}^t$ be the transpose of $\vec{m}$. The matrix $\vec{s}=\vec{m}\vec{m}^t$ is a local stem-stem association matrix. Each element $s_{u,v}$ in $\vec{s}$ expresses a correlation $c_{u,v}$ between the stems s_u and s_v namely,*

$$c_{u,v} = \sum_{d_j \in D_l} f_{s_u,j} \times f_{s_v,j} \tag{5.5}$$

The correlation factor $c_{u,v}$ quantifies the absolute frequencies of co-occurrence and is said to be unnormalized. Thus, if we adopt

$$s_{u,v} = c_{u,v} \tag{5.6}$$

then the association matrix $\vec{s}$ is said to be unnormalized. An alternative is to normalize the correlation factor. For instance, if we adopt

$$s_{u,v} = \frac{c_{u,v}}{c_{u,u} + c_{v,v} - c_{u,v}} \tag{5.7}$$

then the association matrix $\vec{s}$ is said to be normalized. The adoption of normalization yields quite distinct associations as discussed below.

Given a local association matrix $\vec{s}$, we can use it to build local association clusters as follows.

Definition *Consider the u-th row in the association matrix $\vec{s}$ (i.e., the row with all the associations for the stem s_u). Let $S_u(n)$ be a function which takes the u-th row and returns the set of n largest values $s_{u,v}$, where v varies over the set of local stems and $v \neq u$. Then $S_u(n)$ defines a local association cluster around the stem s_u. If $s_{u,v}$ is given by equation 5.6, the association cluster is said to be unnormalized. If $s_{u,v}$ is given by equation 5.7, the association cluster is said to be normalized.*

Given a query q, we are normally interested in finding clusters only for the $|q|$ query terms. Further, it is desirable to keep the size of such clusters small. This means that such clusters can be computed efficiently at query time.

Despite the fact that the above clustering procedure adopts stems, it can equally be applied to non-stemmed keywords. The procedure remains unchanged except for the usage of keywords instead of stems. Keyword-based local clustering is equally worthwhile trying because there is controversy over the advantages of using a stemmed vocabulary, as discussed in Chapter 7.

Metric Clusters

Association clusters are based on the frequency of co-occurrence of pairs of terms in documents and do not take into account *where* the terms occur in a document. Since two terms which occur in the same sentence seem more correlated than two terms which occur far apart in a document, it might be worthwhile to factor in the distance between two terms in the computation of their correlation factor. Metric clusters are based on this idea.

Definition *Let the distance $r(k_i, k_j)$ between two keywords k_i and k_j be given by the number of words between them in a same document. If k_i and k_j are in distinct documents we take $r(k_i, k_j) = \infty$. A local stem-stem metric correlation matrix $\vec{s}$ is defined as follows. Each element $s_{u,v}$ of $\vec{s}$ expresses a metric correlation $c_{u,v}$ between the stems s_u and s_v namely,*

$$c_{u,v} = \sum_{k_i \in V(s_u)} \sum_{k_j \in V(s_v)} \frac{1}{r(k_i, k_j)}$$

In this expression, as already defined, $V(s_u)$ and $V(s_v)$ indicate the sets of keywords which have s_u and s_v as their respective stems. Variations of the above expression for $c_{u,v}$ have been reported in the literature (such as $1/r^2(k_i, k_j)$) but the differences in experimental results are not remarkable.

The correlation factor $c_{u,v}$ quantifies absolute inverse distances and is said to be unnormalized. Thus, if we adopt

$$s_{u,v} = c_{u,v} \tag{5.8}$$

then the association matrix $\vec{s}$ is said to be unnormalized. An alternative is to normalize the correlation factor. For instance, if we adopt

$$s_{u,v} = \frac{c_{u,v}}{|V(s_u)| \times |V(s_v)|} \tag{5.9}$$

then the association matrix $\vec{s}$ is said to be normalized.

Given a local metric matrix $\vec{s}$, we can use it to build local metric clusters as follows.

Definition *Consider the u-th row in the metric correlation matrix $\vec{s}$ (i.e., the row with all the associations for the stem s_u). Let $S_u(n)$ be a function which*

takes the u-th row and returns the set of n largest values $s_{u,v}$, where v varies over the set of local stems and $v \neq u$. Then $S_u(n)$ defines a local metric cluster around the stem s_u. If $s_{u,v}$ is given by equation 5.8, the metric cluster is said to be unnormalized. If $s_{u,v}$ is given by equation 5.9, the metric cluster is said to be normalized.

Scalar Clusters

One additional form of deriving a synonymity relationship between two local stems (or terms) s_u and s_v is by comparing the sets $S_u(n)$ and $S_v(n)$. The idea is that two stems with similar *neighborhoods* have some synonymity relationship. In this case we say that the relationship is indirect or induced by the neighborhood. One way of quantifying such neighborhood relationships is to arrange all correlation values $s_{u,i}$ in a vector $\vec{s}_u$, to arrange all correlation values $s_{v,i}$ in another vector $\vec{s}_v$, and to compare these vectors through a scalar measure. For instance, the cosine of the angle between the two vectors is a popular scalar similarity measure.

Definition *Let $\vec{s}_u = (s_{u,1}, s_{u,2}, \ldots, s_{u,n})$ and $\vec{s}_v = (s_{v,1}, s_{v,2}, \ldots, s_{v,n})$ be two vectors of correlation values for the stems s_u and s_v. Further, let $\vec{s} = (s_{u,v})$ be a scalar association matrix. Then, each $s_{u,v}$ can be defined as*

$$s_{u,v} = \frac{\vec{s}_u \cdot \vec{s}_v}{|\vec{s}_u| \times |\vec{s}_v|} \tag{5.10}$$

The correlation matrix $\vec{s}$ is said to be induced by the neighborhood. Using it, a scalar cluster is then defined as follows.

Definition *Let $S_u(n)$ be a function which returns the set of n largest values $s_{u,v}$, $v \neq u$, defined according to equation 5.10. Then, $S_u(n)$ defines a scalar cluster around the stem s_u.*

Interactive Search Formulation

Stems (or terms) that belong to clusters associated to the query stems (or terms) can be used to expand the original query. Such stems are called neighbors (of the query stems) and are characterized as follows.

A stem s_u which belongs to a cluster (of size n) associated to another stem s_v (i.e., $s_u \in S_v(n)$) is said to be a *neighbor* of s_v. Sometimes, s_u is also called a *searchonym* of s_v but here we opt for using the terminology *neighbor*. While neighbor stems are said to have a synonymity relationship, they are not necessarily synonyms in the grammatical sense. Often, neighbor stems represent distinct keywords which are though correlated by the current query context. The local aspect of this correlation is reflected in the fact that the documents and stems considered in the correlation matrix are all local (i.e., $d_j \in D_l$, $s_u \in V_l$).

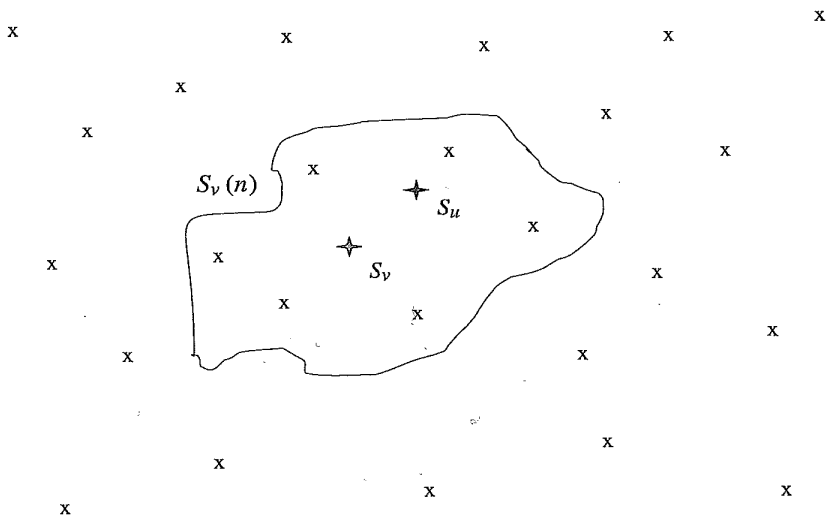


Figure 5.1 Stem s_u as a neighbor of the stem s_v.

Figure 5.1 illustrates a stem (or term) s_u which is located within a neighborhood $S_v(n)$ associated with the stem (or term) s_v. In its broad meaning, neighbor stems are an important product of the local clustering process since they can be used for extending a search formulation in a promising unexpected direction, rather than merely complementing it with missing synonyms.

Consider the problem of expanding a given user query q with neighbor stems (or terms). One possibility is to expand the query as follows. For each stem $s_v \in q$, select m neighbor stems from the cluster $S_v(n)$ (which might be of type association, metric, or scalar) and add them to the query. Hopefully, the additional neighbor stems will retrieve new relevant documents. To cover a broader neighborhood, the set $S_v(n)$ might be composed of stems obtained using correlation factors (i.e., $c_{u,v}$) normalized and unnormalized. The qualitative interpretation is that an unnormalized cluster tends to group stems whose ties are due to their large frequencies, while a normalized cluster tends to group stems which are more rare. Thus, the union of the two clusters provides a better representation of the possible correlations.

Besides the merging of normalized and unnormalized clusters, one can also use information about correlated stems to improve the search. For instance, as before, let two stems s_u and s_v be correlated with a correlation factor $c_{u,v}$. If $c_{u,v}$ is larger than a predefined threshold then a neighbor stem of s_u can also be interpreted as a neighbor stem of s_v and vice versa. This provides greater flexibility, particularly with Boolean queries. To illustrate, consider the expression $(s_u + s_v)$ where the + symbol stands for disjunction. Let $s_{u'}$ be a neighbor stem of s_u. Then, one can try both $(s_{u'} + s_v)$ and $(s_u + s_{u'})$ as synonym search expressions, because of the correlation given by $c_{u,v}$.

Experimental results reported in the literature usually support the hypothesis of the usefulness of local clustering methods. Furthermore, metric clusters seem to perform better than purely association clusters. This strengthens the hypothesis that there is a correlation between the association of two terms and the distance between them.

We emphasize that all the qualitative arguments in this section are explicitly based on the fact that all the clusters are local (i.e., derived solely from the documents retrieved for the current query). In a global context, clusters are derived from all the documents in the collection which implies that our qualitative argumentation might not stand. The main reason is that correlations valid in the whole corpora might not be valid for the current query.

5.3.2 Query Expansion Through Local Context Analysis

The local clustering techniques discussed above are based on the set of documents retrieved for the original query and use the top ranked documents for clustering neighbor terms (or stems). Such a clustering is based on term (stems were considered above) co-occurrence inside documents. Terms which are the best neighbors of each query term are then used to expand the original query q. A distinct approach is to search for term correlations in the whole collection — an approach called global analysis. Global techniques usually involve the building of a thesaurus which identifies term relationships in the whole collection. The terms are treated as concepts and the thesaurus is viewed as a concept relationship structure. Thesauri are expensive to build but, besides providing support for query expansion, are useful as a browsing tool as demonstrated by some search engines in the Web. The building of a thesaurus usually considers the use of small contexts and phrase structures instead of simply adopting the context provided by a whole document. Furthermore, with modern variants of global analysis, terms which are closest to the whole query (and not to individual query terms) are selected for query expansion. The application of ideas from global analysis (such as small contexts and phrase structures) to the local set of documents retrieved is a recent idea which we now discuss.

Local context analysis [838] combines global and local analysis and works as follows. First, the approach is based on the use of noun groups (i.e., a single noun, two adjacent nouns, or three adjacent nouns in the text), instead of simple keywords, as document concepts. For query expansion, concepts are selected from the top ranked documents (as in local analysis) based on their co-occurrence with query terms (no stemming). However, instead of documents, passages (i.e., a text window of fixed size) are used for determining co-occurrence (as in global analysis).

More specifically, the local context analysis procedure operates in three steps.

- First, retrieve the top n ranked passages using the original query. This is accomplished by breaking up the documents initially retrieved by the

query in fixed length passages (for instance, of size 300 words) and ranking these passages as if they were documents.

- Second, for each concept c in the top ranked passages, the similarity $sim(q, c)$ between the whole query q (not individual query terms) and the concept c is computed using a variant of tf-idf ranking.
- Third, the top m ranked concepts (according to $sim(q, c)$) are added to the original query q. To each added concept is assigned a weight given by $1 - 0.9 \times i/m$ where i is the position of the concept in the final concept ranking. The terms in the original query q might be stressed by assigning a weight equal to 2 to each of them.

Of these three steps, the second one is the most complex and the one which we now discuss.

The similarity $sim(q, c)$ between each related concept c and the original query q is computed as follows.

$$sim(q, c) = \prod_{k_i \in q} \left(\delta + \frac{\log(f(c, k_i) \times idf_c)}{\log n} \right)^{idf_i}$$

where n is the number of top ranked passages considered. The function $f(c, k_i)$ quantifies the correlation between the concept c and the query term k_i and is given by

$$f(c, k_i) = \sum_{j=1}^{n} pf_{i,j} \times pf_{c,j}$$

where $pf_{i,j}$ is the frequency of term k_i in the j-th passage and $pf_{c,j}$ is the frequency of the concept c in the j-th passage. Notice that this is the standard correlation measure defined for association clusters (by Equation 5.5) but adapted for passages. The inverse document frequency factors are computed as

$$idf_i = max(1, \frac{\log_{10} N/np_i}{5})$$
$$idf_c = max(1, \frac{\log_{10} N/np_c}{5})$$

where N is the number of passages in the collection, np_i is the number of passages containing the term k_i, and np_c is the number of passages containing the concept c. The factor δ is a constant parameter which avoids a value equal to zero for $sim(q, c)$ (which is useful, for instance, if the approach is to be used with probabilistic frameworks such as that provided by belief networks). Usually, δ is a small factor with values close to 0.1 (10% of the maximum of 1). Finally, the idf_i factor in the exponent is introduced to emphasize infrequent query terms.

The procedure above for computing $sim(q, c)$ is a non-trivial variant of tf-idf ranking. Furthermore, it has been adjusted for operation with TREC data

and did not work so well with a different collection. Thus, it is important to have in mind that tuning might be required for operation with a different collection.

We also notice that the correlation measure adopted with local context analysis is of type association. However, we already know that a correlation of type metric is expected to be more effective. Thus, it remains to be tested whether the adoption of a metric correlation factor (for the function $f(c, k_i)$) makes any difference with local context analysis.

5.4 Automatic Global Analysis

The methods of local analysis discussed above extract information from the local set of documents retrieved to expand the query. It is well accepted that such a procedure yields improved retrieval performance with various collections. An alternative approach is to expand the query using information from the whole set of documents in the collection. Strategies based on this idea are called global analysis procedures. Until the beginning of the 1990s, global analysis was considered to be a technique which failed to yield consistent improvements in retrieval performance with general collections. This perception has changed with the appearance of modern procedures for global analysis. In the following, we discuss two of these modern variants. Both of them are based on a thesaurus-like structure built using all the documents in the collection. However, the approach taken for building the thesaurus and the procedure for selecting terms for query expansion are quite distinct in the two cases.

5.4.1 Query Expansion based on a Similarity Thesaurus

In this section we discuss a query expansion model based on a global similarity thesaurus which is constructed automatically [655]. The similarity thesaurus is based on term to term relationships rather than on a matrix of co-occurrence (as discussed in section 5.3). The distinction is made clear in the discussion below. Furthermore, special attention is paid to the selection of terms for expansion and to the reweighting of these terms. In contrast to previous global analysis approaches, terms for expansion are selected based on their similarity to the whole query rather than on their similarities to individual query terms.

A *similarity thesaurus* is built considering term to term relationships. However, such relationships are not derived directly from co-occurrence of terms inside documents. Rather, they are obtained by considering that the terms are concepts in a concept space. In this concept space, each term is indexed by the documents in which it appears. Thus, terms assume the original role of documents while documents are interpreted as indexing elements. The following definitions establish the proper framework.

Definition *As before (see Chapter 2), let t be the number of terms in the collection, N be the number of documents in the collection, and $f_{i,j}$ be the frequency*

of occurrence of the term k_i in the document d_j. Further, let t_j be the number of distinct index terms in the document d_j and itf_j be the inverse term frequency for document d_j. Then,

$$itf_j = \log \frac{t}{t_j}$$

analogously to the definition of inverse document frequency.

Within this framework, to each term k_i is associated a vector $\vec{k}_i$ given by

$$\vec{k}_i = (w_{i,1}, w_{i,2}, \ldots, w_{i,N})$$

where, as in Chapter 2, $w_{i,j}$ is a weight associated to the index-document pair $[k_i, d_j]$. Here, however, these weights are computed in a rather distinct form as follows.

$$w_{i,j} = \frac{(0.5 + 0.5\frac{f_{i,j}}{max_j(f_{i,j})})\ \ itf_j}{\sqrt{\sum_{l=1}^{N}(0.5 + 0.5\frac{f_{i,l}}{max_l(f_{i,l})})^2\ itf_j^2}} \tag{5.11}$$

where $max_j(f_{i,j})$ computes the maximum of all factors $f_{i,j}$ for the i-th term (i.e., over all documents in the collection). We notice that the expression above is a variant of tf-idf weights but one which considers inverse term frequencies instead.

The relationship between two terms k_u and k_v is computed as a correlation factor $c_{u,v}$ given by

$$c_{u,v} = \vec{k}_u \cdot \vec{k}_v = \sum_{\forall\ d_j} w_{u,j} \times w_{v,j} \tag{5.12}$$

We notice that this is a variation of the correlation measure used for computing scalar association matrices (defined by Equation 5.5). The main difference is that the weights are based on interpreting documents as indexing elements instead of repositories for term co-occurrence. The global similarity thesaurus is built through the computation of the correlation factor $c_{u,v}$ for each pair of indexing terms $[k_u, k_v]$ in the collection (analogously to the procedure in section 5.3). Of course, this is computationally expensive. However, this global similarity thesaurus has to be computed only once and can be updated incrementally.

Given the global similarity thesaurus, query expansion is done in three steps as follows.

- First, represent the query in the concept space used for representation of the index terms.

- Second, based on the global similarity thesaurus, compute a similarity $sim(q, k_v)$ between each term k_v correlated to the query terms and the whole query q.
- Third, expand the query with the top r ranked terms according to $sim(q, k_v)$.

For the first step, the query is represented in the concept space of index term vectors as follows.

Definition *To the query q is associated a vector $\vec{q}$ in the term-concept space given by*

$$\vec{q} = \sum_{k_i \in q} w_{i,q} \vec{k}_i$$

where $w_{i,q}$ is a weight associated to the index-query pair $[k_i, q]$. This weight is computed analogously to the index-document weight formula in equation 5.11.

For the second step, a similarity $sim(q, k_v)$ between each term k_v (correlated to the query terms) and the user query q is computed as

$$sim(q, k_v) = \vec{q} \cdot \vec{k}_v = \sum_{k_u \in Q} w_{u,q} \times c_{u,v}$$

where $c_{u,v}$ is the correlation factor given in equation 5.12. As illustrated in Figure 5.2, a term might be quite close to the whole query while its distances to individual query terms are larger. This implies that the terms selected here for query expansion might be distinct from those selected by previous global analysis methods (which adopted a similarity to individual query terms for deciding terms for query expansion).

For the third step, the top r ranked terms according to $sim(q, k_v)$ are added to the original query q to form the expanded query q'. To each expansion term k_v in the query q' is assigned a weight $w_{v,q'}$ given by

$$w_{v,q'} = \frac{sim(q, k_v)}{\sum_{k_u \in q} w_{u,q}}$$

The expanded query q' is then used to retrieve new documents to the user. This completes the technique for query expansion based on a similarity thesaurus. Contrary to previous global analysis approaches, this technique has yielded improved retrieval performance (in the range of 20%) with three different collections.

It is worthwhile making one final observation. Consider a document d_j which is represented in the term-concept space by $\vec{d}_j = \sum_{k_i \in d_j} w_{i,j} \vec{k}_i$. Further, assume that the original query q is expanded to include all the t index terms

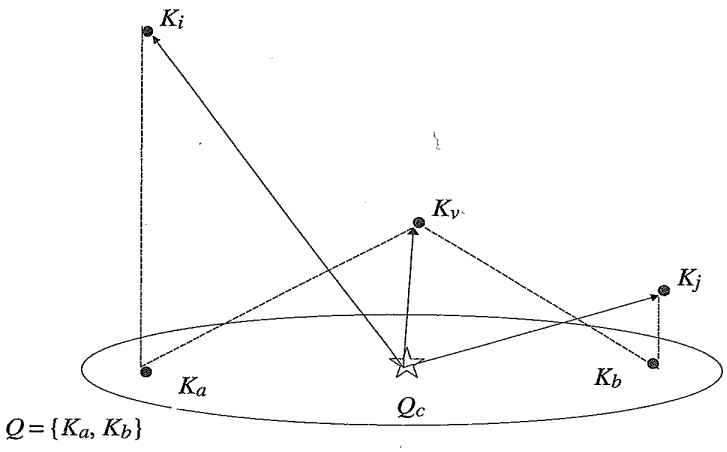


Figure 5.2 The distance of a given term k_v to the query centroid Q_c might be quite distinct from the distances of k_v to the individual query terms.

(properly weighted) in the collection. Then, the similarity $sim(q, d_j)$ between the document d_j and the query q can be computed in the term-concept space by

$$sim(q, d_j) \quad \alpha \sum_{k_v \in d_j} \sum_{k_u \in q} w_{v,j} \times w_{u,q} \times c_{u,v}$$

Such an expression is analogous to the formula for query-document similarity in the generalized vector space model (see Chapter 2). Thus, the generalized vector space model can be interpreted as a query expansion technique. The main differences with the term-concept idea are the weight computation and the fact that only the top r ranked terms are used for query expansion with the term-concept technique.

5.4.2 Query Expansion based on a Statistical Thesaurus

In this section, we discuss a query expansion technique based on a global statistical thesaurus [200]. Despite also being a global analysis technique, the approach is quite distinct from the one described above which is based on a similarity thesaurus.

The global thesaurus is composed of classes which group correlated terms in the context of the whole collection. Such correlated terms can then be used to expand the original user query. To be effective, the terms selected for expansion must have high term discrimination values [699] which implies that they must be low frequency terms. However, it is difficult to cluster low frequency terms effectively due to the small amount of information about them (they occur in few documents). To circumvent this problem, we cluster documents into

classes instead and use the low frequency terms in these documents to define our thesaurus classes. In this situation, the document clustering algorithm must produce small and tight clusters.

A document clustering algorithm which produces small and tight clusters is the *complete link algorithm* which works as follows (naive formulation).

(1) Initially, place each document in a distinct cluster.
(2) Compute the similarity between all pairs of clusters.
(3) Determine the pair of clusters $[C_u, C_v]$ with the highest inter-cluster similarity.
(4) Merge the clusters C_u and C_v.
(5) Verify a stop criterion. If this criterion is not met then go back to step 2.
(6) Return a hierarchy of clusters.

The similarity between two clusters is defined as the minimum of the similarities between all pairs of inter-cluster documents (i.e., two documents not in the same cluster). To compute the similarity between documents in a pair, the cosine formula of the vector model is used. As a result of this minimality criterion, the resultant clusters tend to be small and tight.

Consider that the whole document collection has been clustered using the complete link algorithm. Figure 5.3 illustrates a small portion of the whole cluster hierarchy in which $sim(C_u, C_v) = 0.15$ and $sim(C_{u+v}, C_z) = 0.11$ where C_{u+v} is a reference to the cluster which results from merging C_u and C_v. Notice that the similarities decrease as we move up in the hierarchy because high level clusters include more documents and thus represent a looser grouping. Thus, the tightest clusters lie at the bottom of the clustering hierarchy.

Given the document cluster hierarchy for the whole collection, the terms that compose each class of the global thesaurus are selected as follows.

- Obtain from the user three parameters: threshold class (TC), number of

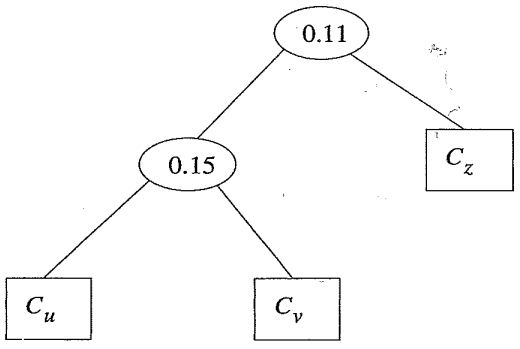


Figure 5.3 Hierarchy of three clusters (inter-cluster similarities indicated in the ovals) generated by the complete link algorithm.

documents in a class (NDC), and minimum inverse document frequency (MIDF).

- Use the parameter TC as a threshold value for determining the document clusters that will be used to generate thesaurus classes. This threshold has to be surpassed by $sim(C_u, C_v)$ if the documents in the clusters C_u and C_v are to be selected as sources of terms for a thesaurus class. For instance, in Figure 5.3, a value of 0.14 for TC returns the thesaurus class C_{u+v} while a value of 0.10 for TC returns the classes C_{u+v} and C_{u+v+z}.
- Use the parameter NDC as a limit on the size of clusters (number of documents) to be considered. For instance, if both C_{u+v} and C_{u+v+z} are preselected (through the parameter TC) then the parameter NDC might be used to decide between the two. A low value of NDC might restrict the selection to the smaller cluster C_{u+v}.
- Consider the set of documents in each document cluster preselected above (through the parameters TC and NDC). Only the lower frequency documents are used as sources of terms for the thesaurus classes. The parameter MIDF defines the minimum value of inverse document frequency for any term which is selected to participate in a thesaurus class. By doing so, it is possible to ensure that only *low frequency* terms participate in the thesaurus classes generated (terms too generic are not good synonyms).

Given that the thesaurus classes have been built, they can be used for query expansion. For this, an average term weight wt_C for each thesaurus class C is computed as follows.

$$wt_C = \frac{\sum_{i=1}^{|C|} w_{i,C}}{|C|}$$

where $|C|$ is the number of terms in the thesaurus class C and $w_{i,C}$ is a precomputed weight associated with the term-class pair $[k_i, C]$. This average term weight can then be used to compute a thesaurus class weight w_C as

$$w_C = \frac{wt_C}{|C|} \times 0.5$$

The above weight formulations have been verified through experimentation and have yielded good results.

Experiments with four test collections (ADI, Medlars, CACM, and ISI; see Chapter 3 for details on these collections) indicate that global analysis using a thesaurus built by the complete link algorithm might yield consistent improvements in retrieval performance.

The main problem with this approach is the initialization of the parameters TC, NDC, and MIDF. The threshold value TC depends on the collection and can be difficult to set properly. Inspection of the cluster hierarchy is almost always necessary for assisting with the setting of TC. Care must be exercised because a

high value of TC might yield classes with too few terms while a low value of TC might yield too few classes. The selection of the parameter NDC can be decided more easily once TC has been set. However, the setting of the parameter MIDF might be difficult and also requires careful consideration.

5.5 Trends and Research Issues

The relevance feedback strategies discussed here can be directly applied to the graphical interfaces of modern information systems. However, since interactivity is now of greater importance, new techniques for capturing feedback information from the user are desirable. For instance, there is great interest in graphical interfaces which display the documents in the answer set as points in a 2D or 3D space. The motivation is to allow the user to quickly identify (by visual inspection) relationships among the documents in the answer. In this scenario, a rather distinct strategy for quantifying feedback information might be required. Thus, relevance strategies for dealing with visual displays are an important research problem.

In the past, global analysis was viewed as an approach which did not yield good improvements in retrieval performance. However, new results obtained at the beginning of the 1990s changed this perception. Further, the Web has provided evidence that techniques based on global analysis might be of interest to the users. For instance, this is the case with the highly popular 'Yahoo!' software which uses a manually built hierarchy of concepts to assist the user with forming the query. This suggests that investigating the utilization of global analysis techniques in the Web is a promising research problem.

Local analysis techniques are interesting because they take advantage of the local context provided with the query. In this regard, they seem more appropriate than global analysis techniques. Furthermore, many positive results have been reported in the literature. The application of local analysis techniques to the Web, however, has not been explored and is a promising research direction. The main challenge is the computational burden imposed on the search engine site due to the need to process document texts at query time. Thus, a related research problem of relevance is the development of techniques for speeding up query processing at the search engine site. In truth, this problem is of interest even if one considers only the normal processing of the queries because the search engines depend on processing as many queries as possible for economic survival.

The combination of local analysis, global analysis, visual displays, and interactive interfaces is also a current and important research problem. Allowing the user to visually explore the document space and providing him with clues which assist with the query formulation process are highly relevant issues. Positive results in this area might become a turning point regarding the design of user interfaces and are likely to attract wide attention.

5.6 Bibliographic Discussion

Query expansion methods have been studied for a long time. While the success of expansion methods throughout the years has been debatable, more recently researchers have reached the consensus that query expansion is a useful and little explored (commercially) technique. Useful because its modern variants can be used to consistently improve the retrieval performance with general collections. Little explored because few commercial systems (and Web search engines) take advantage of it.

Early work suggesting the expansion of a user query with closely related terms was done by Maron and Kuhns in 1960 [547]. The classic technique for combining query expansion with term reweighting in the vector model was studied by Rocchio in 1965 (using the Smart system [695] as a testbed) and published later on [678]. Ide continued the studies of Rocchio and proposed variations to the term reweighting formula [391].

The probabilistic model was introduced by Robertson and Sparck Jones [677] in 1976. A thorough and entertaining discussion of this model can be found in the book by van Rijsbergen [785]. Croft and Harper [199] suggested that the initial search should use a distinct computation. In 1983, Croft [198] proposed to extend the probabilistic formula to include within-document frequencies and introduced the C and K parameters.

Since the probabilistic model does not provide means of expanding the query, query expansion has to be done separately. In 1978, Harper and van Rijsbergen [345] used a term-term clustering technique based on a maximum spanning tree to select terms for probabilistic query expansion. Two years later, they also introduced a new relevance weighting scheme, called EMIM [344], to be used with their query expansion technique. In 1981, Wu and Salton [835] used relevance feedback to reweight terms (using a probabilistic formula) extracted from relevant documents and used these terms to expand the query. Empirical results showed improvements in retrieval performance.

Our discussion on user relevance feedback for the vector and probabilistic models in section 5.2 is based on four sources: a nice paper by Salton and Buckley [696], the book by van Rijsbergen [785], the book by Salton and McGill [698], and two book chapters by Harman [340, 339].

Regarding automatic query expansion, Lesk [500] tried variations of term-term clustering in the Smart system without positive results. Following that, Sparck Jones and Barber [413] and Minker, Wilson and Zimmerman [562] also observed no improvements with query expansion based on term-term global clustering. These early research results left the impression that query expansion based on global analysis is not an effective technique. However, more recent results show that this is not the case. In fact, the research results obtained by Vorhees [793], by Crouch and Yang [200], and by Qiu and Frei [655] indicate that query expansion based on global analysis can consistently yield improved retrieval performance.

Our discussion on query expansion through local clustering is based on early work by Attar and Fraenkel [35] from 1977. The idea of local context

analysis is much more recent and was introduced by Xu and Croft [838] in 1996. The discussion on query expansion using a global similarity thesaurus is based on the work by Qiu and Frei [655]. Finally, the discussion on query expansion using a global statistical thesaurus is based on the work of Crouch and Yang [200] which is influenced by the term discrimination value theory introduced by Salton, Yang, and Yu [699] early in 1975.

Since query expansion frequently is based on some form of clustering, our discussion covered a few clustering algorithms. However, our aim was not to provide a thorough review of clustering algorithms for information retrieval. Such a review can be found in the work of Rasmussen [668].

Chapter 8
Indexing and Searching

with Gonzalo Navarro

8.1 Introduction

Chapter 4 describes the query operations that can be performed on text databases. In this chapter we cover the main techniques we need to implement those query operations.

We first concentrate on searching queries composed of words and on reporting the documents where they are found. The number of occurrences of a query in each document and even its exact positions in the text may also be required. Following that, we concentrate on algorithms dealing with Boolean operations. We then consider sequential search algorithms and pattern matching. Finally, we consider structured text and compression techniques.

An obvious option in searching for a basic query is to scan the text sequentially. Sequential or online text searching involves finding the occurrences of a pattern in a text when the text is not preprocessed. Online searching is appropriate when the text is small (i.e., a few megabytes), and it is the only choice if the text collection is very volatile (i.e., undergoes modifications very frequently) or the index space overhead cannot be afforded.

A second option is to build data structures over the text (called *indices*) to speed up the search. It is worthwhile building and maintaining an index when the text collection is large and *semi-static*. Semi-static collections can be updated at reasonably regular intervals (e.g., daily) but they are not deemed to support thousands of insertions of single words per second, say. This is the case for most real text databases, not only dictionaries or other slow growing literary works. For instance, it is the case for Web search engines or journal archives.

Nowadays, the most successful techniques for medium size databases (say up to 200Mb) combine online and indexed searching.

We cover three main indexing techniques: inverted files, suffix arrays, and signature files. Keyword-based search is discussed first. We emphasize inverted files, which are currently the best choice for most applications. Suffix trees

and arrays are faster for phrase searches and other less common queries, but are harder to build and maintain. Finally, signature files were popular in the 1980s, but nowadays inverted files outperform them. For all the structures we pay attention not only to their search cost and space overhead, but also to the cost of building and updating them.

We assume that the reader is familiar with basic data structures, such as sorted arrays, binary search trees, B-trees, hash tables, and tries. Since tries are heavily used we give a brief and simplified reminder here. Tries, or digital search trees, are multiway trees that store sets of strings and are able to retrieve any string in time proportional to its length (independent of the number of strings stored). A special character is added to the end of the string to ensure that no string is a prefix of another. Every edge of the tree is labeled with a letter. To search a string in a trie, one starts at the root and scans the string character-wise, descending by the appropriate edge of the trie. This continues until a leaf is found (which represents the searched string) or the appropriate edge to follow does not exist at some point (i.e., the string is not in the set). See Figure 8.3 for an example of a text and a trie built on its words.

Although an index must be built prior to searching it, we present these tasks in the reverse order. We think that understanding first how a data structure is used makes it clear how it is organized, and therefore eases the understanding of the construction algorithm, which is usually more complex.

Throughout this chapter we make the following assumptions. We call n the size of the text database. Whenever a pattern is searched, we assume that it is of length m, which is much smaller than n. We call M the amount of main memory available. We assume that the modifications which a text database undergoes are additions, deletions, and replacements (which are normally made by a deletion plus an addition) of pieces of text of size $n' < n$.

We give experimental measures for many algorithms to give the reader a grasp of the real times involved. To do this we use a reference architecture throughout the chapter, which is representative of the power of today's computers. We use a 32-bit Sun UltraSparc-1 of 167 MHz with 64 Mb of RAM, running Solaris. The code is written in C and compiled with all optimization options. For the text data, we use collections from TREC-2, specifically WSJ, DOE, FR, ZIFF and AP. These are described in more detail in Chapter 3.

8.2 Inverted Files

An inverted file (or inverted index) is a word-oriented mechanism for indexing a text collection in order to speed up the searching task. The inverted file structure is composed of two elements: the *vocabulary* and the *occurrences*. The vocabulary is the set of all different words in the text. For each such word a list of all the text positions where the word appears is stored. The set of all those lists is called the 'occurrences' (Figure 8.1 shows an example). These positions can refer to words or characters. Word positions (i.e., position i refers to the i-th word) simplify

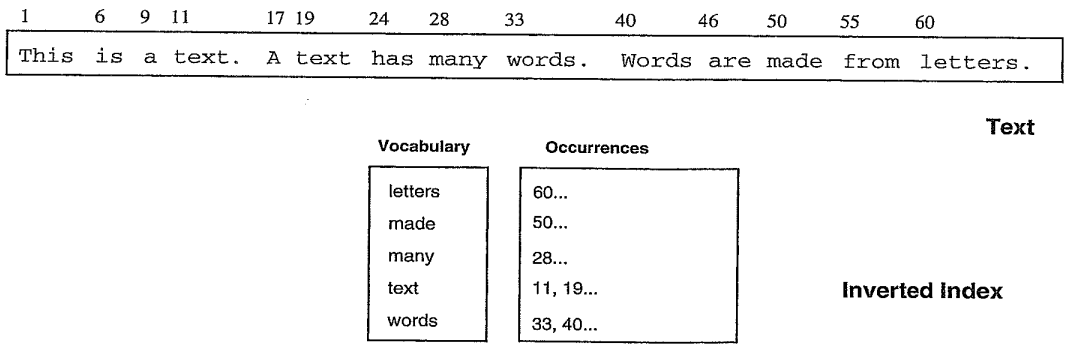


Figure 8.1 A sample text and an inverted index built on it. The words are converted to lower-case and some are not indexed. The occurrences point to character positions in the text.

phrase and proximity queries, while character positions (i.e., the position i is the i-th character) facilitate direct access to the matching text positions.

Some authors make the distinction between inverted files and inverted lists. In an inverted file, each element of a list points to a document or file name, while inverted lists match our definition. We prefer not to make such a distinction because, as we will see later, this is a matter of the *addressing granularity*, which can range from text positions to logical blocks.

The space required for the vocabulary is rather small. According to Heaps' law (see Chapter 6) the vocabulary grows as $O(n^\beta)$, where β is a constant between 0 and 1 dependent on the text, being between 0.4 and 0.6 in practice. For instance, for 1 Gb of the TREC-2 collection the vocabulary has a size of only 5 Mb. This may be further reduced by stemming and other normalization techniques as described in Chapter 7.

The occurrences demand much more space. Since each word appearing in the text is referenced once in that structure, the extra space is $O(n)$. Even omitting stopwords (which is the default practice when words are indexed), in practice the space overhead of the occurrences is between 30% and 40% of the text size.

To reduce space requirements, a technique called *block addressing* is used. The text is divided in blocks, and the occurrences point to the blocks where the word appears (instead of the exact positions). The classical indices which point to the exact occurrences are called 'full inverted indices.' By using block addressing not only can the pointers be smaller because there are fewer blocks than positions, but also all the occurrences of a word inside a single block are collapsed to one reference (see Figure 8.2). Indices of only 5% overhead over the text size are obtained with this technique. The price to pay is that, if the exact occurrence positions are required (for instance, for a proximity query), then an online search over the qualifying blocks has to be performed. For instance, block addressing indices with 256 blocks stop working well with texts of 200 Mb.

Table 8.1 presents the projected space taken by inverted indices for texts of

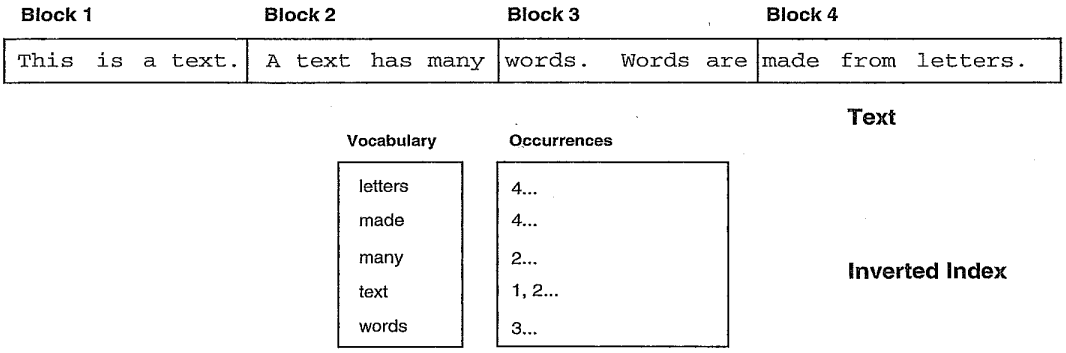


Figure 8.2 The sample text split into four blocks, and an inverted index using block addressing built on it. The occurrences denote block numbers. Notice that both occurrences of 'words' collapsed into one.

different sizes, with and without the use of stopwords. The full inversion stands for inverting all the words and storing their exact positions, using four bytes per pointer. The document addressing index assumes that we point to documents which are of size 10 Kb (and the necessary number of bytes per pointer, i.e. one, two, and three bytes, depending on text size). The block addressing index assumes that we use 256 or 64K blocks (one or two bytes per pointer) independently of the text size. The space taken by the pointers can be significantly reduced by using compression. We assume that 45% of all the words are stopwords, and that there is one non-stopword each 11.5 characters. Our estimation for the vocabulary is based on Heaps' law with parameters $V = 30n^{0.5}$. All these decisions were taken according to our experience and experimentally validated.

The blocks can be of fixed size (imposing a logical block structure over the text database) or they can be defined using the natural division of the text collection into files, documents, Web pages, or others. The division into blocks of fixed size improves efficiency at retrieval time, i.e. the more variance in the block sizes, the more amount of text sequentially traversed on average. This is because larger blocks match queries more frequently and are more expensive to traverse.

Alternatively, the division using natural cuts may eliminate the need for online traversal. For example, if one block per retrieval unit is used and the exact match positions are not required, there is no need to traverse the text for single-word queries, since it is enough to know which retrieval units to report. But if, on the other hand, many retrieval units are packed into a single block, the block has to be traversed to determine which units to retrieve.

It is important to notice that in order to use block addressing, the text must be readily available at search time. This is not the case for remote text (as in Web search engines), or if the text is in a CD-ROM that has to be mounted, for instance. Some restricted queries not needing exact positions can still be solved if the blocks are retrieval units.

Index	Small collection (1 Mb)		Medium collection (200 Mb)		Large collection (2 Gb)	
Addressing words	45%	73%	36%	64%	35%	63%
Addressing documents	19%	26%	18%	32%	26%	47%
Addressing 64K blocks	27%	41%	18%	32%	5%	9%
Addressing 256 blocks	18%	25%	1.7%	2.4%	0.5%	0.7%

Table 8.1 Sizes of an inverted file as approximate percentages of the size the whole text collection. Four granularities and three collections are considered. For each collection, the right column considers that stopwords are not indexed while the left column considers that all words are indexed.

8.2.1 Searching

The search algorithm on an inverted index follows three general steps (some may be absent for specific queries):

- **Vocabulary search** The words and patterns present in the query are isolated and searched in the vocabulary. Notice that phrases and proximity queries are split into single words.
- **Retrieval of occurrences** The lists of the occurrences of all the words found are retrieved.
- **Manipulation of occurrences** The occurrences are processed to solve phrases, proximity, or Boolean operations. If block addressing is used it may be necessary to directly search the text to find the information missing from the occurrences (e.g., exact word positions to form phrases).

Hence, searching on an inverted index always starts in the vocabulary. Because of this it is a good idea to have it in a separate file. It is possible that this file fits in main memory even for large text collections.

Single-word queries can be searched using any suitable data structure to speed up the search, such as hashing, tries, or B-trees. The first two give $O(m)$ search cost (independent of the text size). However, simply storing the words in lexicographical order is cheaper in space and very competitive in performance, since the word can be binary searched at $O(\log n)$ cost. Prefix and range queries can also be solved with binary search, tries, or B-trees, but not with hashing. If the query is formed by single words, then the process ends by delivering the list of occurrences (we may need to make a union of many lists if the pattern matches many words).

Context queries are more difficult to solve with inverted indices. Each element must be searched separately and a list (in increasing positional order) generated for each one. Then, the lists of all elements are traversed in synchronization to find places where all the words appear in sequence (for a phrase) or appear close enough (for proximity). If one list is much shorter than the others, it may be better to binary search its elements into the longer lists instead of performing a linear merge. It is possible to prove using Zipf's law that this is normally the case. This is important because the most time-demanding operation on inverted indices is the merging or intersection of the lists of occurrences.

If the index stores character positions the phrase query cannot allow the separators to be disregarded, and the proximity has to be defined in terms of character distance.

Finally, note that if block addressing is used it is necessary to traverse the blocks for these queries, since the position information is needed. It is then better to intersect the lists to obtain the blocks which contain all the searched words and then sequentially search the context query in those blocks as explained in section 8.5. Some care has to be exercised at block boundaries, since they can split a match. This part of the search, if present, is also quite time consuming.

Using Heaps' and the generalized Zipf's laws, it has been demonstrated that the cost of solving queries is sublinear in the text size, even for complex queries involving list merging. The time complexity is $O(n^{\alpha})$, where α depends on the query and is close to 0.4..0.8 for queries with reasonable selectivity.

Even if block addressing is used and the blocks have to be traversed, it is possible to select the block size as an increasing function of n, so that not only does the space requirement keep sublinear but also the amount of text traversed in all useful queries is also sublinear.

Practical figures show, for instance, that both the space requirement and the amount of text traversed can be close to $O(n^{0.85})$. Hence, inverted indices allow us to have sublinear search time at sublinear space requirements. This is not possible on the other indices.

Search times on our reference machine for a full inverted index built on 250 Mb of text give the following results: searching a simple word took 0.08 seconds, while searching a phrase took 0.25 to 0.35 seconds (from two to five words).

8.2.2 Construction

Building and maintaining an inverted index is a relatively low cost task. In principle, an inverted index on a text of n characters can be built in $O(n)$ time. All the vocabulary known up to now is kept in a trie data structure, storing for each word a list of its occurrences (text positions). Each word of the text is read and searched in the trie. If it is not found, it is added to the trie with an empty list of occurrences. Once it is in the trie, the new position is added to the end of its list of occurrences. Figure 8.3 illustrates this process.

Once the text is exhausted, the trie is written to disk together with the list of occurrences. It is good practice to split the index into two files. In the

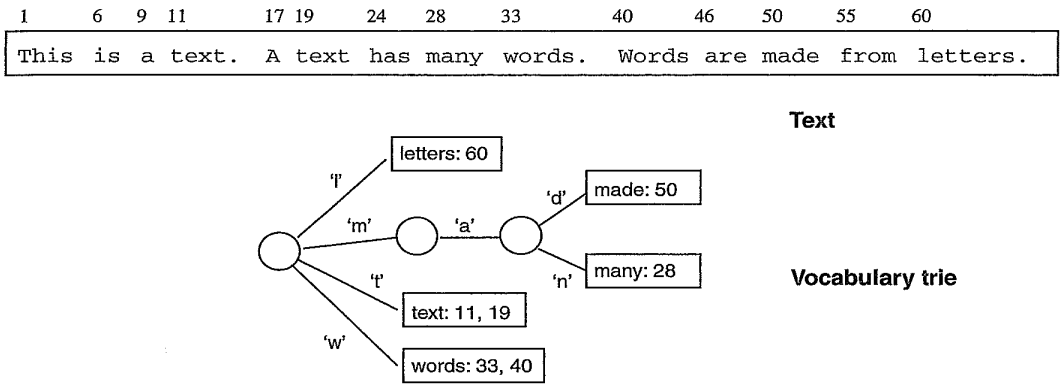


Figure 8.3 Building an inverted index for the sample text.

first file, the lists of occurrences are stored contiguously. In this scheme, the file is typically called a 'posting file'. In the second file, the vocabulary is stored in lexicographical order and, for each word, a pointer to its list in the first file is also included. This allows the vocabulary to be kept in memory at search time in many cases. Further, the number of occurrences of a word can be immediately known from the vocabulary with little or no space overhead.

We analyze now the construction time under this scheme. Since in the trie $O(1)$ operations are performed per text character, and the positions can be inserted at the end of the lists of occurrences in $O(1)$ time, the overall process is $O(n)$ worst-case time.

However, the above algorithm is not practical for large texts where the index does not fit in main memory. A paging mechanism will severely degrade the performance of the algorithm. We describe an alternative which is faster in practice.

The algorithm already described is used until the main memory is exhausted (if the trie takes up too much space it can be replaced by a hash table or other structure). When no more memory is available, the *partial* index I_i obtained up to now is written to disk and erased from main memory before continuing with the rest of the text.

Finally, a number of partial indices I_i exist on disk. These indices are then merged in a hierarchical fashion. Indices I_1 and I_2 are merged to obtain the index $I_{1..2}$; I_3 and I_4 produce $I_{3..4}$; and so on. The resulting partial indices are now approximately twice the size. When all the indices at this level have been merged in this way, the merging proceeds at the next level, joining the index $I_{1..2}$ with the index $I_{3..4}$ to form $I_{1..4}$. This is continued until there is just one index comprising the whole text, as illustrated in Figure 8.4.

Merging two indices consists of merging the sorted vocabularies, and whenever the same word appears in both indices, merging both lists of occurrences. By construction, the occurrences of the smaller-numbered index are before those of the larger-numbered index, and therefore the lists are just concatenated. This is a very fast process in practice, and its complexity is $O(n_1 + n_2)$, where n_1 and n_2 are the sizes of the indices.

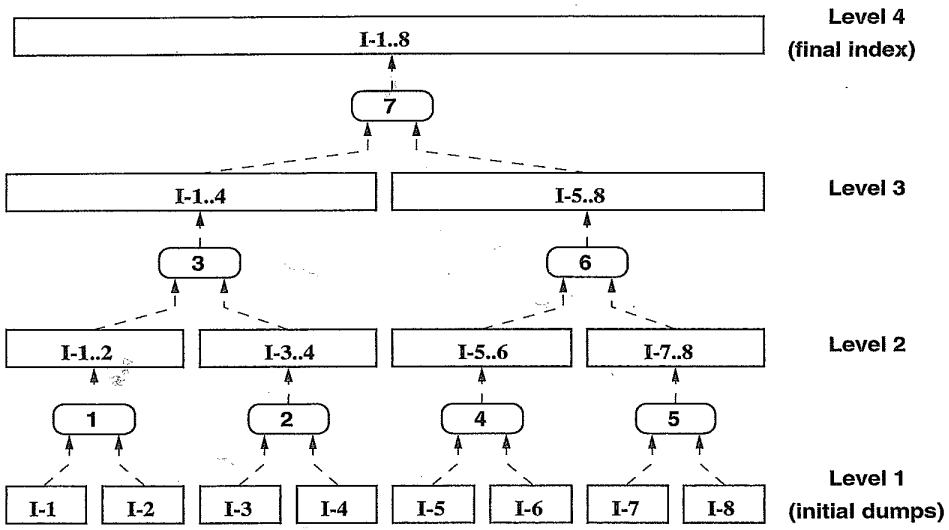


Figure 8.4 Merging the partial indices in a binary fashion. Rectangles represent partial indices, while rounded rectangles represent merging operations. The numbers inside the merging operations show a possible merging order.

The total time to generate the partial indices is $O(n)$ as before. The number of partial indices is $O(n/M)$. Each level of merging performs a linear process over the whole index (no matter how it is split into partial indices at this level) and thus its cost is $O(n)$. To merge the $O(n/M)$ partial indices, $\log_2(n/M)$ merging levels are necessary, and therefore the cost of this algorithm is $O(n \log(n/M))$.

More than two indices can be merged at once. Although this does not change the complexity, it improves efficiency since fewer merging levels exist. On the other hand, the memory buffers for each partial index to merge will be smaller and hence more disk seeks will be performed. In practice it is a good idea to merge even 20 partial indices at once.

Real times to build inverted indices on the reference machine are between 4-8 Mb/min for collections of up to 1 Gb (the slowdown factor as the text grows is barely noticeable). Of this time, 20-30% is spent on merging the partial indices.

To reduce build-time space requirements, it is possible to perform the merging in-place. That is, when two or more indices are merged, write the result in the same disk blocks of the original indices instead of on a new file. It is also a good idea to perform the hierarchical merging as soon as the files are generated (e.g., collapse I_1 and I_2 into $I_{1..2}$ as soon as I_2 is produced). This also reduces space requirements because the vocabularies are merged and redundant words are eliminated (there is no redundancy in the occurrences). The vocabulary can be a significative part of the smaller partial indices, since they represent a small text.

This algorithm changes very little if block addressing is used. Index maintenance is also cheap. Assume that a new text of size n' is added to the database. The inverted index for the new text is built and then both indices are merged

as is done for partial indices. This takes $O(n + n' \log(n'/M))$. Deleting text can be done by an $O(n)$ pass over the index eliminating the occurrences that point inside eliminated text areas (and eliminating words if their lists of occurrences disappear in the process).

8.3 Other Indices for Text

8.3.1 Suffix Trees and Suffix Arrays

Inverted indices assume that the text can be seen as a sequence of words. This restricts somewhat the kinds of queries that can be answered. Other queries such as phrases are expensive to solve. Moreover, the concept of word does not exist in some applications such as genetic databases.

In this section we present suffix arrays. Suffix arrays are a space efficient implementation of suffix trees. This type of index allows us to answer efficiently more complex queries. Its main drawbacks are its costly construction process, that the text must be readily available at query time, and that the results are not delivered in text position order. This structure can be used to index only words (without stopwords) as the inverted index as well as to index any text character. This makes it suitable for a wider spectrum of applications, such as genetic databases. However, for word-based applications, inverted files perform better unless complex queries are an important issue.

This index sees the text as one long string. Each position in the text is considered as a text *suffix* (i.e., a string that goes from that text position to the end of the text). It is not difficult to see that two suffixes starting at different positions are lexicographically different (assume that a character smaller than all the rest is placed at the end of the text). Each suffix is thus uniquely identified by its position.

Not all text positions need to be indexed. *Index points* are selected from the text, which point to the *beginning* of the text positions which will be retrievable. For instance, it is possible to index only word beginnings to have a functionality similar to inverted indices. Those elements which are not index points are not retrievable (as in an inverted index it is not possible to retrieve the middle of a word). Figure 8.5 illustrates this.

Structure

In essence, a suffix tree is a trie data structure built over all the suffixes of the text. The pointers to the suffixes are stored at the leaf nodes. To improve space utilization, this trie is compacted into a Patricia tree. This involves compressing unary paths, i.e. paths where each node has just one child. An indication of the next character position to consider is stored at the nodes which root a compressed path. Once unary paths are not present the tree has $O(n)$ nodes instead of the worst-case $O(n^2)$ of the trie (see Figure 8.6).

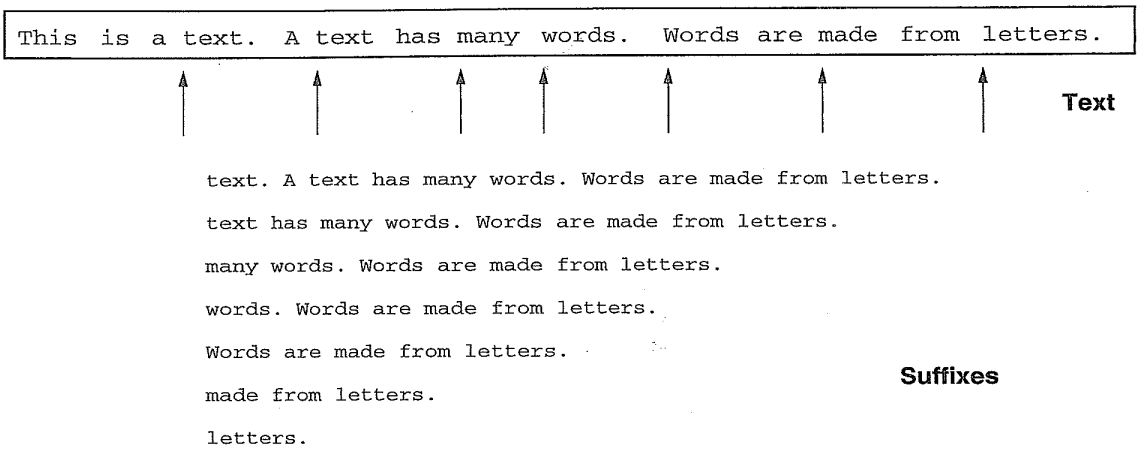


Figure 8.5 The sample text with the index points of interest marked. Below, the suffixes corresponding to those index points.

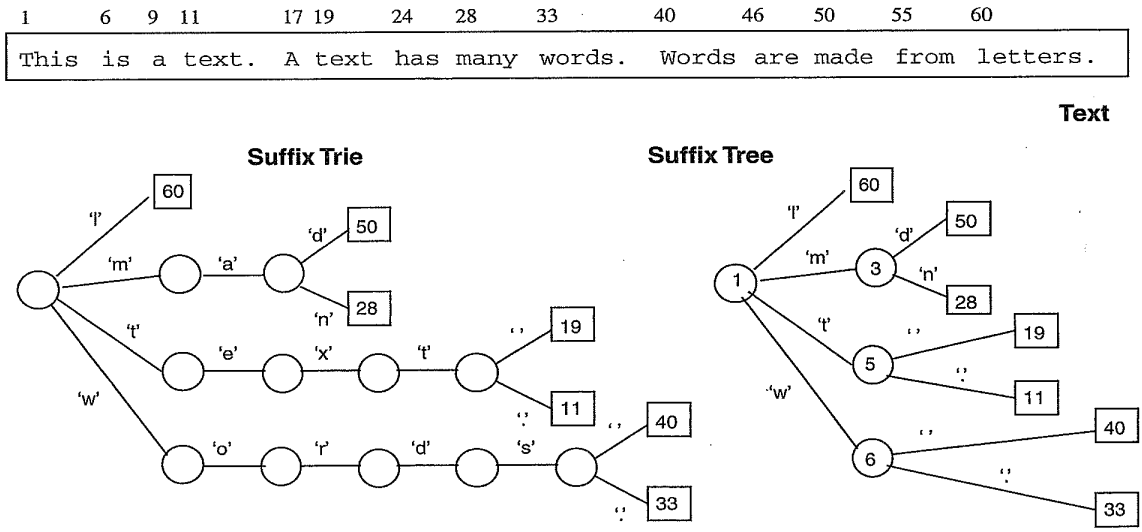


Figure 8.6 The suffix trie and suffix tree for the sample text.

The problem with this structure is its space. Depending on the implementation, each node of the trie takes 12 to 24 bytes, and therefore even if only word beginnings are indexed, a space overhead of 120% to 240% over the text size is produced.

Suffix arrays provide essentially the same functionality as suffix trees with much less space requirements. If the leaves of the suffix tree are traversed in left-to-right order (top to bottom in our figures), all the suffixes of the text are retrieved in lexicographical order. A suffix array is simply an array containing all the pointers to the text suffixes listed in lexicographical order, as shown in Figure 8.7. Since they store one pointer per indexed suffix, the space requirements

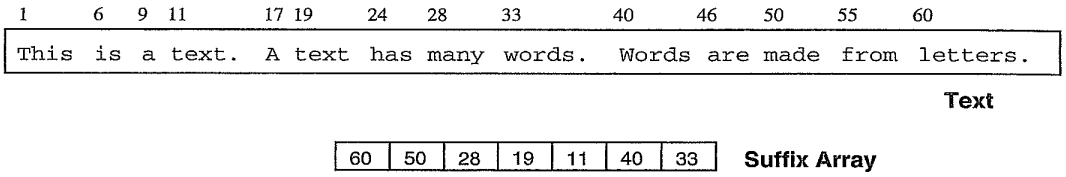


Figure 8.7 The suffix array for the sample text.

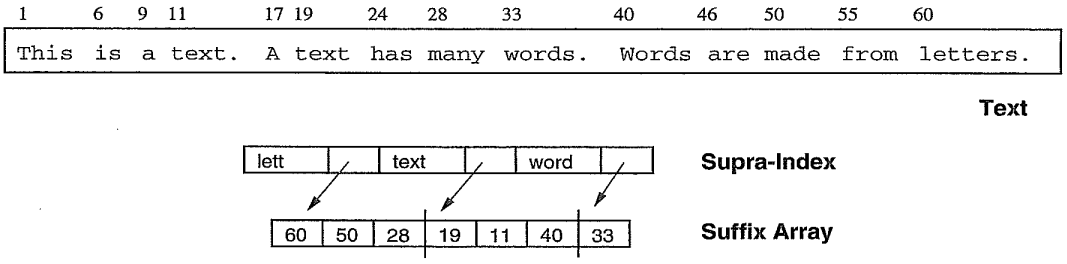


Figure 8.8 A supra-index over our suffix array. One out of three entries are sampled, keeping their first four characters. The pointers (arrows) are in fact unnecessary.

are almost the same as those for inverted indices (disregarding compression techniques), i.e. close to 40% overhead over the text size.

Suffix arrays are designed to allow binary searches done by comparing the contents of each pointer. If the suffix array is large (the usual case), this binary search can perform poorly because of the number of random disk accesses. To remedy this situation, the use of *supra-indices* over the suffix array has been proposed. The simplest supra-index is no more than a sampling of one out of b suffix array entries, where for each sample the first ℓ suffix characters are stored in the supra-index. This supra-index is then used as a first step of the search to reduce external accesses. Figure 8.8 shows an example.

This supra-index does not in fact need to take samples at fixed intervals, nor to take samples of the same length. For word-indexing suffix arrays it has been suggested that a new sample could be taken each time the first word of the suffix changes, and to store the word instead of ℓ characters. This is exactly the same as having a vocabulary of the text plus pointers to the array. In fact, the only important difference between this structure and an inverted index is that the occurrences of each word in an inverted index are sorted by text position, while in a suffix array they are sorted lexicographically by the text following the word. Figure 8.9 illustrates this relationship.

The extra space requirements of supra-indices are modest. In particular, it is clear that the space requirements of the suffix array with a vocabulary supra-index are exactly the same as for inverted indices (except for compression, as we see later).

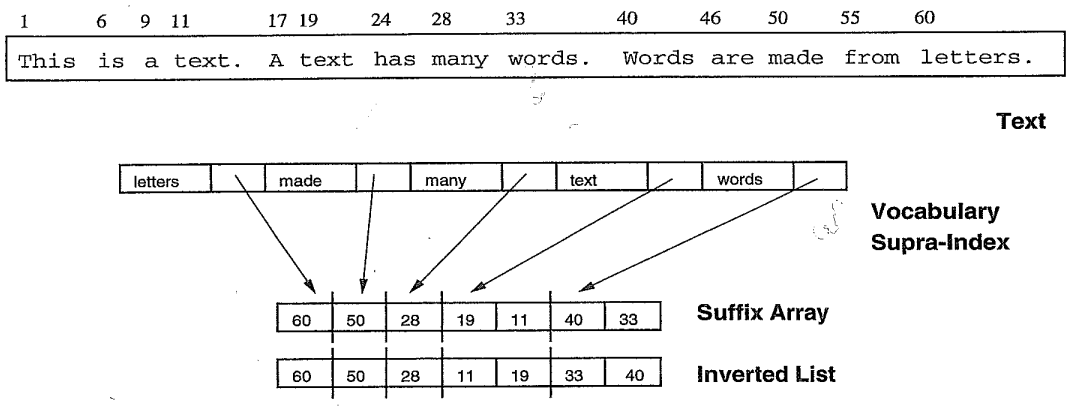


Figure 8.9 Relationship between our inverted list and suffix array with vocabulary supra-index.

Searching

If a suffix tree on the text can be afforded, many basic patterns such as words, prefixes, and phrases can be searched in $O(m)$ time by a simple trie search. However, suffix trees are not practical for large texts, as explained. Suffix arrays, on the other hand, can perform the same search operations in $O(\log n)$ time by doing a binary search instead of a trie search.

This is achieved as follows: the search pattern originates two 'limiting patterns' P_1 and P_2, so that we want any suffix S such that $P_1 \leq S < P_2$. We binary search both limiting patterns in the suffix array. Then, all the elements lying between both positions point to exactly those suffixes that start like the original pattern (i.e., to the pattern positions in the text). For instance, in our example of figure 8.9, in order to find the word 'text' we search for 'text' and 'texu', obtaining the portion of the array that contains the pointers 19 and 11.

All these queries retrieve a subtree of the suffix tree or an interval of the suffix array. The results have to be collected later, which may imply sorting them in ascending text order. This is a complication of suffix trees or arrays with respect to inverted indices.

Simple phrase searching is a good case for these indices. A simple phrase of words can be searched as if it was a simple pattern. This is because the suffix tree/array sorts with respect to the complete suffixes and not only their first word. A proximity search, on the other hand, has to be solved element-wise. The matches for each element must be collected and sorted and then they have to be intersected as for inverted files.

The binary search performed on suffix arrays, unfortunately, is done on disk, where the accesses to (random) text positions force a seek operation which spans the disk tracks containing the text. Since a random seek is $O(n)$ in size, this makes the search cost $O(n \log n)$ time. Supra-indices are used as a first step in any binary search operation to alleviate this problem. To avoid performing $O(\log n)$ random accesses to the text on disk (and to the suffix array on disk), the search starts in the supra-index, which usually fits in main memory (text samples

included). After this search is completed, the suffix array block which is between the two selected samples is brought into memory and the binary search is completed (performing random accesses to the text on disk). This reduces disk search times to close to 25% of the original time. Modified binary search techniques that sacrifice the exact partition in the middle of the array taking into account the current disk head position allow a further reduction from 40% to 60%.

Search times in a 250 Mb text in our reference machine are close to 1 second for a simple word or phrase, while the part corresponding to the accesses to the text sums up 0.6 seconds. The use of supra-indices should put the total time close to 0.3 seconds. Note that the times, although high for simple words, do not degrade for long phrases as with inverted indices.

Construction in Main Memory

A suffix tree for a text of n characters can be built in $O(n)$ time. The algorithm, however, performs poorly if the suffix tree does not fit in main memory, which is especially stringent because of the large space requirements of the suffix trees. We do not cover the linear algorithm here because it is quite complex and only of theoretical interest.

We concentrate on direct suffix array construction. Since the suffix array is no more than the set of pointers lexicographically sorted, the pointers are collected in ascending text order and then just sorted by the text they point to. Note that in order to compare two suffix array entries the corresponding text positions must be accessed. These accesses are basically random. Hence, both the suffix array and the text must be in main memory. This algorithm costs $O(n \log n)$ string comparisons.

An algorithm to build the suffix array in $O(n \log n)$ character comparisons follows. All the suffixes are bucket-sorted in $O(n)$ time according to the first letter only. Then, each bucket is bucket-sorted again, now according to their first two letters. At iteration i, the suffixes begin already sorted by their 2^{i-1} first letters and end up sorted by their first 2^i letters. As at each iteration the total cost of all the bucket sorts is $O(n)$, the total time is $O(n \log n)$, and the average is $O(n \log \log n)$ (since $O(\log n)$ comparisons are necessary on average to distinguish two suffixes of a text). This algorithm accesses the text only in the first stage (bucket sort for the first letter).

In order to sort the strings in the i-th iteration, notice that since *all* suffixes are sorted by their first 2^{i-1} letters, to sort the text positions $T_{a\ldots}$ and $T_{b\ldots}$ in the suffix array (assuming that they are in the same bucket, i.e., they share their first 2^{i-1} letters), it is enough to determine the relative order between text positions $T_{a+2^{i-1}}$ and $T_{b+2^{i-1}}$ in the current stage of the search. This can be done in constant time by storing the reverse permutation. We do not enter here into further detail.

Construction of Suffix Arrays for Large Texts

There is still the problem that large text databases will not fit in main memory. It could be possible to apply an external memory sorting algorithm. However,

each comparison involves accessing the text at random positions on the disk. This will severely degrade the performance of the sorting process.

We explain an algorithm especially designed for large texts. Split the text into blocks that can be sorted in main memory. Then, for each block, build its suffix array in main memory and merge it with the rest of the array already built for the previous text. That is:

- build the suffix array for the first block,
- build the suffix array for the second block,
- merge both suffix arrays,
- build the suffix array for the third block,
- merge the new suffix array with the previous one,
- build the suffix array for the fourth block,
- merge the new suffix array with the previous one,
- ... and so on.

The difficult part is how to merge a large suffix array (already built) with the small suffix array (just built). The merge needs to compare text positions which are spread in a large text, so the problem persists. The solution is to first determine how many elements of the large array are to be placed between each pair of elements in the small array, and later use that information to merge the arrays without accessing the text. Hence, the information that we need is how many suffixes of the large text lie between each pair of positions of the small suffix array. We compute counters that store this information.

The counters are computed without using the large suffix array. The text corresponding to the large array is sequentially read into main memory. Each suffix of that text is *searched* in the small suffix array (in main memory). Once we find the inter-element position where the suffix lies, we just increment the appropriate counter. Figure 8.10 illustrates this process.

We analyze this algorithm now. If there is $O(M)$ main memory to index, then there will be $O(n/M)$ text blocks. Each block is merged against an array of size $O(n)$, where all the $O(n)$ suffixes of the large text are binary searched in the small suffix array. This gives a total CPU complexity of $O(n^2 \log(M)/M)$.

Notice that this same algorithm can be used for index maintenance. If a new text of size n' is added to the database, it can be split into blocks as before and merged block-wise into the current suffix array. This will take $O(nn' \log(M)/M)$. To delete some text it suffices to perform an $O(n)$ pass over the array eliminating all the text positions which lie in the deleted areas.

As can be seen, the construction process is in practice more costly for suffix arrays than for inverted files. The construction of the supra-index consists of a fast final sequential pass over the suffix array.

Indexing times for 250 Mb of text are close to 0.8 Mb/min on the reference machine. This is five to ten times slower than the construction of inverted indices.

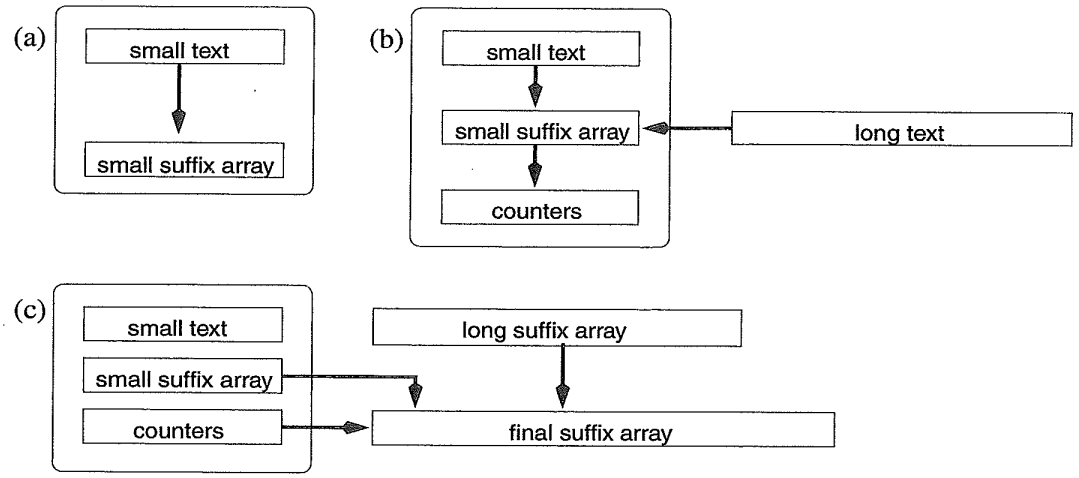


Figure 8.10 A step of the suffix array construction for large texts: (*a*) the local suffix array is built, (*b*) the counters are computed, (*c*) the suffix arrays are merged.

8.3.2 Signature Files

Signature files are word-oriented index structures based on hashing. They pose a low overhead (10% to 20% over the text size), at the cost of forcing a sequential search over the index. However, although their search complexity is linear (instead of sublinear as with the previous approaches), its constant is rather low, which makes the technique suitable for not very large texts. Nevertheless, inverted files outperform signature files for most applications.

Structure

A signature file uses a hash function (or 'signature') that maps words to bit masks of B bits. It divides the text in blocks of b words each. To each text block of size b, a bit mask of size B will be assigned. This mask is obtained by bitwise ORing the signatures of all the words in the text block. Hence, the signature file is no more than the sequence of bit masks of all blocks (plus a pointer to each block). The main idea is that if a word is present in a text block, then all the bits set in its signature are also set in the bit mask of the text block. Hence, whenever a bit is set in the mask of the query word and not in the mask of the text block, then the word is not present in the text block. Figure 8.11 shows an example.

However, it is possible that all the corresponding bits are set even though the word is not there. This is called a *false drop*. The most delicate part of the design of a signature file is to ensure that the probability of a false drop is low enough while keeping the signature file as short as possible.

The hash function is forced to deliver bit masks which have at least ℓ bits set. A good model assumes that ℓ bits are randomly set in the mask (with possible repetition). Let $\alpha = \ell/B$. Since each of the b words sets ℓ bits at

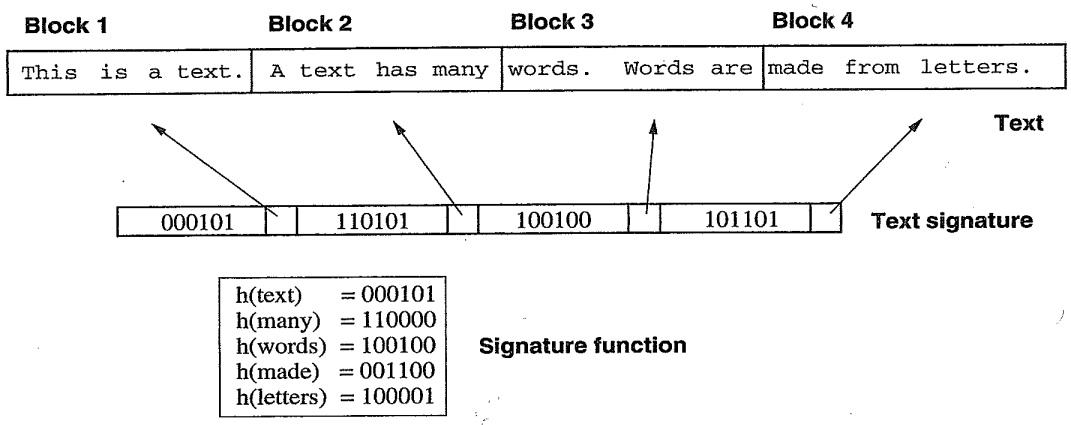


Figure 8.11 A signature file for our sample text cut into blocks.

random, the probability that a given bit of the mask is set in a word signature is $1-(1-1/B)^{b\ell} \approx 1-e^{-b\alpha}$. Hence, the probability that the ℓ random bits set in the query are also set in the mask of the text block is

$$(1-e^{-b\alpha})^{\alpha B}$$

which is minimized for $\alpha = \ln(2)/b$. The false drop probability under the optimal selection $\ell = B\ln(2)/b$ is $(1/2^{\ln(2)})^{B/b} = 1/2^{\ell}$.

Hence, a reasonable proportion B/b must be determined. The space overhead of the index is approximately $(1/80) \times (B/b)$ because B is measured in bits and b in words. Then, the false drop probability is a function of the overhead to pay. For instance, a 10% overhead implies a false drop probability close to 2%, while a 20% overhead errs with probability 0.046%. This error probability corresponds to the expected amount of sequential searching to perform while checking if a match is a false drop or not.

Searching

Searching a single word is carried out by hashing it to a bit mask W, and then comparing the bit masks B_i of all the text blocks. Whenever $(W \ \& \ B_i = W)$, where & is the bitwise AND, all the bits set in W are also set in B_i and therefore the text block *may* contain the word. Hence, for all candidate text blocks, an online traversal must be performed to verify if the word is actually there. This traversal cannot be avoided as in inverted files (except if the risk of a false drop is accepted).

No other types of patterns can be searched in this scheme. On the other hand, the scheme is more efficient to search phrases and reasonable proximity queries. This is because *all* the words must be present in a block in order for that block to hold the phrase or the proximity query. Hence, the bitwise OR of all the query masks is searched, so that *all* their bits must be present. This

reduces the probability of false drops. This is the only indexing scheme which improves in phrase searching.

Some care has to be exercised at block boundaries, however, to avoid missing a phrase which crosses a block limit. To allow searching phrases of j words or proximities of up to j words, consecutive blocks must overlap in j words.

If the blocks correspond to retrieval units, simple Boolean conjunctions involving words or phrases can also be improved by forcing all the relevant words to be in the block.

We were only able to find real performance estimates from 1992, run on a Sun 3/50 with local disk. Queries on a small 2.8 Mb database took 0.42 seconds. Extrapolating to today's technology, we find that the performance should be close to 20 Mb/sec (recall that it is linear time), and hence the example of 250 Mb of text would take 12 seconds, which is quite slow.

Construction

The construction of a signature file is rather easy. The text is simply cut in blocks, and for each block an entry of the signature file is generated. This entry is the bitwise OR of the signatures of all the words in the block.

Adding text is also easy, since it is only necessary to keep adding records to the signature file. Text deletion is carried out by deleting the appropriate bit masks.

Other storage proposals exist apart from storing all the bit masks in sequence. For instance, it is possible to make a different file for each bit of the mask, i.e. one file holding all the first bits, another file for all the second bits, etc. This reduces the disk times to search for a query, since only the files corresponding to the ℓ bits which are set in the query have to be traversed.

8.4 Boolean Queries

We now cover set manipulation algorithms. These algorithms are used when operating on sets of results, which is the case in Boolean queries. Boolean queries are described in Chapter 4, where the concept of *query syntax tree* is defined.

Once the leaves of the query syntax tree are solved (using the algorithms to find the documents containing the basic queries given), the relevant documents must be worked on by composition operators. Normally the search proceeds in three phases: the first phase determines which documents classify, the second determines the relevance of the classifying documents so as to present them appropriately to the user, and the final phase retrieves the exact positions of the matches to highlight them in those documents that the user actually wants to see.

This scheme avoids doing unnecessary work on documents which will not classify at last (first phase), or will not be read at last (second phase). However, some phases can be merged if doing the extra operations is not expensive. Some phases may not be present at all in some scenarios.

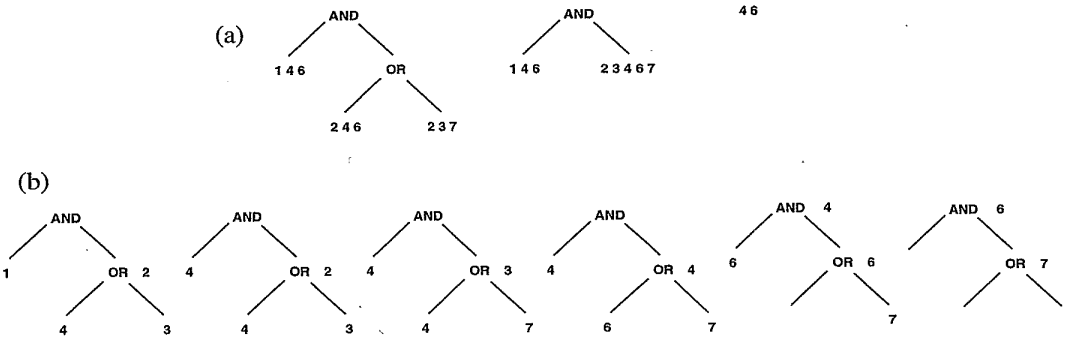


Figure 8.12 Processing the internal nodes of the query syntax tree. In (*a*) full evaluation is used. In (*b*) we show lazy evaluation in more detail.

Once the leaves of the query syntax tree find the classifying sets of documents, these sets are further operated by the internal nodes of the tree. It is possible to algebraically optimize the tree using identities such as $a\ OR\ (a\ AND\ b) = a$, for instance, or sharing common subexpressions, but we do not cover this issue here.

As all operations need to pair the same document in both their operands, it is good practice to keep the sets sorted, so that operations like intersection, union, etc. can proceed sequentially on both lists and also generate a sorted list. Other representations for sets not consisting of the list of matching documents (such as bit vectors) are also possible.

Under this scheme, it is possible to evaluate the syntax tree in *full* or *lazy* form. In the full evaluation form, both operands are first completely obtained and then the complete result is generated. In lazy evaluation, results are delivered only when required, and to obtain that result some data is recursively required to both operands.

Full evaluation allows some optimizations to be performed because the sizes of the results are known in advance (for instance, merging a very short list against a very long one can proceed by binary searching the elements of the short list in the long one). Lazy evaluation, on the other hand, allows the application to control when to do the work of obtaining new results, instead of blocking it for a long time. Hybrid schemes are possible, for example obtain all the leaves at once and then proceed in lazy form. This may be useful, for instance, to implement some optimizations or to ensure that all the accesses to the index are sequential (thus reducing disk seek times). Figure 8.12 illustrates this.

The complexity of solving these types of queries, apart from the cost of obtaining the results at the leaves, is normally linear in the total size of all the intermediate results. This is why this time may dominate the others, when there are huge intermediate results. This is more noticeable to the user when the final result is small.

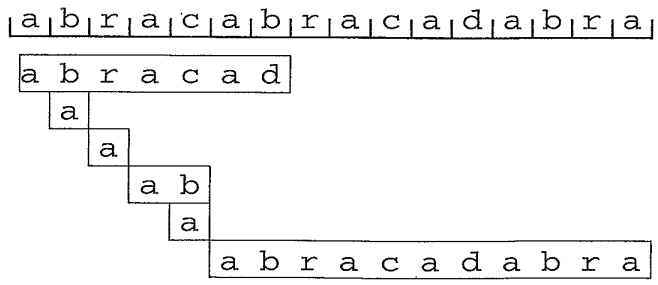


Figure 8.13 Brute-force search algorithm for the pattern '`abracadabra`.' Squared areas show the comparisons performed.

8.5 Sequential Searching

We now cover the algorithms for text searching when no data structure has been built on the text. As shown, this is a basic part of some indexing techniques as well as the only option in some cases. We cover exact string matching in this section. Later we cover matching of more complex patterns. Our exposition is mainly conceptual and the implementation details are not shown (see the Bibliographic Discussion at the end of this chapter for more information).

The problem of exact string matching is: given a short pattern P of length m and a long text T of length n, find all the text positions where the pattern occurs. With minimal changes this problem subsumes many basic queries, such as word, prefix, suffix, and substring search.

This is a classical problem for which a wealth of solutions exists. We sketch the main algorithms, and leave aside a lot of the theoretical work that is not competitive in practice. For example, we do not include the Karp-Rabin algorithm, which is a nice application of hashing to string searching, but is not practical. We also briefly cover multipattern algorithms (that search many patterns at once), since a query may have many patterns and it may be more efficient to retrieve them all at once. Finally, we also mention how to do phrases and proximity searches.

We assume that the text and the pattern are sequences of characters drawn from an alphabet of size σ, whose first character is at position 1. The average-case analysis assumes random text and patterns.

8.5.1 Brute Force

The brute-force (BF) algorithm is the simplest possible one. It consists of merely trying all possible pattern positions in the text. For each such position, it verifies whether the pattern matches at that position. See Figure 8.13.

Since there are $O(n)$ text positions and each one is examined at $O(m)$ worst-case cost, the worst-case of brute-force searching is $O(mn)$. However, its average

case is $O(n)$ (since on random text a mismatch is found after $O(1)$ comparisons on average). This algorithm does not need any pattern preprocessing.

Many algorithms use a modification of this scheme. There is a *window* of length m which is slid over the text. It is *checked* whether the text in the window is equal to the pattern (if it is, the window position is reported as a match). Then, the window is *shifted* forward. The algorithms mainly differ in the way they check and shift the window.

8.5.2 Knuth-Morris-Pratt

The KMP algorithm was the first with linear worst-case behavior, although on average it is not much faster than BF. This algorithm also slides a window over the text. However, it does not try all window positions as BF does. Instead, it reuses information from previous checks.

After the window is checked, whether it matched the pattern or not, a number of pattern letters were compared to the text window, and they all matched except possibly the last one compared. Hence, when the window has to be shifted, there is a *prefix* of the pattern that matched the text. The algorithm takes advantage of this information to avoid trying window positions which can be deduced not to match.

The pattern is preprocessed in $O(m)$ time and space to build a table called *next*. The *next* table at position j says which is the longest proper prefix of $P_{1..j-1}$ which is also a suffix and the characters following prefix and suffix are different. Hence $j - next[j] + 1$ window positions can be safely skipped if the characters up to $j-1$ matched, and the j-th did not. For instance, when searching the word 'abracadabra,' if a text window matched up to 'abracab,' five positions can be safely skipped since $next[7] = 1$. Figure 8.14 shows an example.

The crucial observation is that this information depends only on the pattern, because if the text in the window matched up to position $j-1$, then that text is equal to the pattern.

The algorithm moves a window over the text and a pointer inside the window. Each time a character matches, the pointer is advanced (a match is reported if the pointer reaches the end of the window). Each time a character is not matched, the window is shifted forward in the text, to the position given by *next*, but the pointer position in the text does not change. Since at each text comparison the window or the pointer advance by at least one position, the algorithm performs at most $2n$ comparisons (and at least n).

The Aho-Corasick algorithm can be regarded as an extension of KMP in matching a set of patterns. The patterns are arranged in a trie-like data structure. Each trie node represents having matched a prefix of some pattern(s). The *next* function is replaced by a more general set of *failure* transitions. Those transitions go between nodes of the trie. A transition leaving from a node representing the prefix x leads to a node representing a prefix y, such that y is the longest prefix in the set of patterns which is also a proper suffix of x. Figure 8.15 illustrates this.

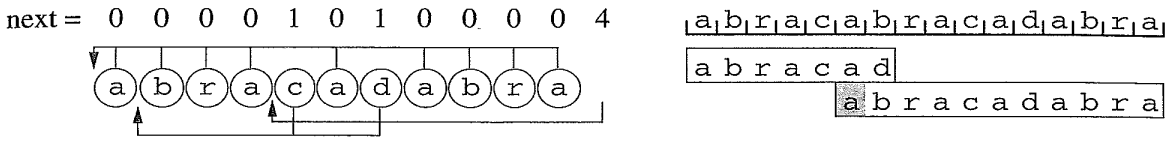


Figure 8.14 KMP algorithm searching '`abracadabra`.' On the left, an illustration of the *next* function. Notice that after matching '`abracada`' we do not try to match the last '`a`' with the first one since what follows cannot be a '`b`.' On the right, a search example. Grayed areas show the prefix information reused.

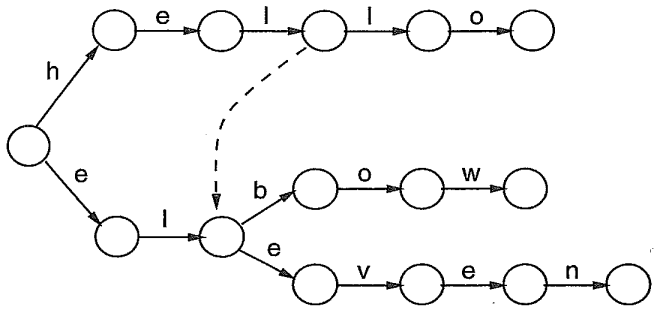


Figure 8.15 Aho-Corasick trie example for the set '`hello`,' '`elbow`' and '`eleven`' showing only one of all the failure transitions.

This trie, together with its failure transitions, is built in $O(m)$ time and space (where m is the total length of all the patterns). Its search time is $O(n)$ no matter how many patterns are searched. Much as KMP, it makes at most $2n$ inspections.

8.5.3 Boyer-Moore Family

BM algorithms are based on the fact that the check inside the window can proceed backwards. When a match or mismatch is determined, a *suffix* of the pattern has been compared and found equal to the text in the window. This can be used in a way very similar to the *next* table of KMP, i.e. compute for every pattern position j the next-to-last occurrence of $P_{j..m}$ inside P. This is called the 'match heuristic.'

This is combined with what is called the 'occurrence heuristic.' It states that the text character that produced the mismatch (if a mismatch occurred) has to be aligned with the same character in the pattern after the shift. The heuristic which gives the longest shift is selected.

For instance, assume that '`abracadabra`' is searched in a text which starts with '`abracababra`.' After matching the suffix '`abra`' the underlined text character '`b`' will cause a mismatch. The match heuristic states that since '`abra`' was matched a shift of 7 is safe. The occurrence heuristic states that since the underlined '`b`' must match the pattern, a shift of 5 is safe. Hence, the pattern is

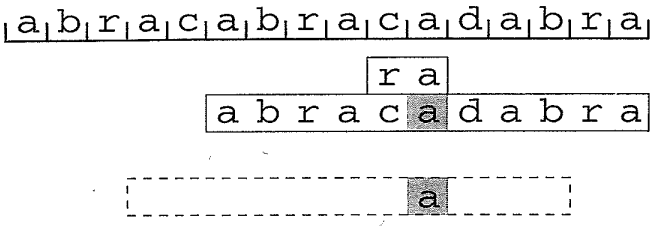


Figure 8.16 BM algorithm searching '`abracadabra`.' Squared areas show the comparisons performed. Grayed areas have already been compared (but the algorithm compares them again). The dashed box shows the match heuristic, which was not chosen.

shifted by 7. See Figure 8.16.

The preprocessing time and space of this algorithm is $O(m+\sigma)$. Its search time is $O(n\log(m)/m)$ on average, which is 'sublinear' in the sense that not all characters are inspected. On the other hand, its worst case is $O(mn)$ (unlike KMP, the old suffix information is not kept to avoid further comparisons).

Further simplifications of the BM algorithm lead to some of the fastest algorithms on average. The Simplified BM algorithm uses only the occurrence heuristic. This obtains almost the same shifts in practice. The BM-Horspool (BMH) algorithm does the same, but it notices that it is not important any more that the check proceeds backwards, and uses the occurrence heuristic on the *last* character of the window instead of the one that caused the mismatch. This gives longer shifts on average. Finally, the BM-Sunday (BMS) algorithm modifies BMH by using the character *following* the last one, which improves the shift especially on short patterns.

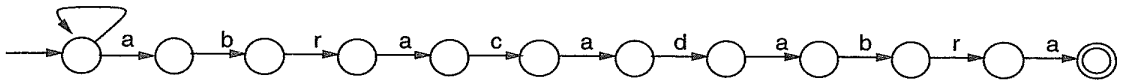
The Commentz-Walter algorithm is an extension of BM to multipattern search. It builds a trie on the reversed patterns, and instead of a backward window check, it enters into the trie with the window characters read backwards. A shift function is computed by a natural extension of BM. In general this algorithm improves over Aho-Corasick for not too many patterns.

8.5.4 Shift-Or

Shift-Or is based on *bit-parallelism*. This technique involves taking advantage of the intrinsic parallelism of the bit operations inside a computer word (of w bits). By cleverly using this fact, the number of operations that an algorithm performs can be cut by a factor of at most w. Since in current architectures w is 32 or 64, the speedup is very significant in practice.

The Shift-Or algorithm uses bit-parallelism to simulate the operation of a non-deterministic automaton that searches the pattern in the text (see Figure 8.17). As this automaton is simulated in time $O(mn)$, the Shift-Or algorithm achieves $O(mn/w)$ worst-case time (optimal speedup).

The algorithm first builds a table B which for each character stores a bit mask $b_m...b_1$. The mask in $B[c]$ has the i-th bit set to zero if and only if



$B[a] =$	1	0	0	1	0	1	0	1	0	0	1
$B[b] =$	0	1	0	0	0	0	0	0	1	0	0
$B[r] =$	0	0	1	0	0	0	0	0	0	1	0
$B[c] =$	0	0	0	0	1	0	0	0	0	0	0
$B[d] =$	0	0	0	0	0	0	1	0	0	0	0
$B[*] =$	0	0	0	0	0	0	0	0	0	0	0

Figure 8.17 Non-deterministic automaton that searches '`abracadabra`,' and the associated B table. The initial self-loop matches any character. Each table column corresponds to an edge of the automaton.

$p_i = c$ (see Figure 8.17). The state of the search is kept in a machine word $D = d_m...d_1$, where d_i is zero whenever the state numbered i in Figure 8.17 is active. Therefore, a match is reported whenever d_m is zero. In the following, we use '|' to denote the bitwise OR and '&' to denote the bitwise AND.

D is set to all ones originally, and for each new text character T_j, D is updated using the formula

$$D' \leftarrow (D << 1) \mid B[T_j]$$

(where '<<' means shifting all the bits in D one position to the left and setting the rightmost bit to zero). It is not hard to relate the formula to the movement that occurs in the non-deterministic automaton for each new text character.

For patterns longer than the computer word (i.e., $m > w$), the algorithm uses $\lceil m/w \rceil$ computer words for the simulation (not all them are active all the time). The algorithm is $O(n)$ on average and the preprocessing is $O(m+\sigma)$ time and $O(\sigma)$ space.

It is easy to extend Shift-Or to handle classes of characters by manipulating the B table and keeping the search algorithm unchanged.

This paradigm also can search a large set of extended patterns, as well as multiple patterns (where the complexity is the same as before if we consider that m is the total length of all the patterns).

8.5.5 Suffix Automaton

The Backward DAWG matching (BDM) algorithm is based on a suffix automaton. A *suffix automaton* on a pattern P is an automaton that recognizes all the suffixes of P. The non-deterministic version of this automaton has a very regular structure and is shown in Figure 8.18.

The BDM algorithm converts this automaton to deterministic. The size and construction time of this automaton is $O(m)$. This is basically the preprocessing effort of the algorithm. Each path from the initial node to any internal

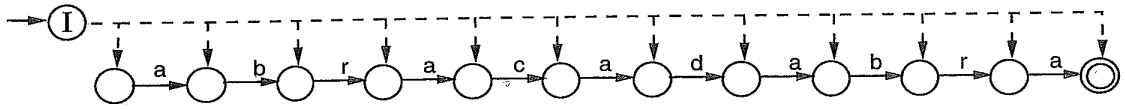


Figure 8.18 A non-deterministic suffix automaton. Dashed lines represent ε-transitions (i.e., they occur without consuming any input). I is the initial state of the automaton.

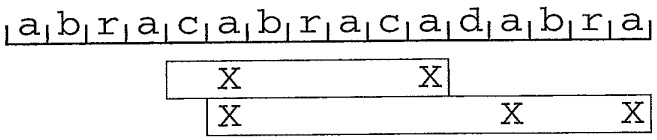


Figure 8.19 The BDM algorithm for the pattern '`abracadabra`.' The rectangles represent elements compared to the text window. The `X`s show the positions where a pattern prefix was recognized.

node represents a substring of the pattern. The final nodes represent pattern suffixes.

To search a pattern P, the suffix automaton of P^r (the reversed pattern) is built. The algorithm searches backwards inside the text window for a substring of the pattern P using the suffix automaton. Each time a terminal state is reached before hitting the beginning of the window, the position inside the window is remembered. This corresponds to finding a *prefix* of the pattern equal to a suffix of the window (since the reverse suffixes of P^r are the prefixes of P). The last prefix recognized backwards is the *longest* prefix of P in the window. A match is found if the complete window is read, while the check is abandoned when there is no transition to follow in the automaton. In either case, the window is shifted to align with the longest prefix recognized. See Figure 8.19.

This algorithm is $O(mn)$ time in the worst case and $O(n \log(m)/m)$ on average. There exists also a multipattern version of this algorithm called MultiBDM, which is the fastest for many patterns or very long patterns.

BDM rarely beats the best BM algorithms. However, a recent bit-parallel implementation called BNDM improves over BM in a wide range of cases. This algorithm simulates the non-deterministic suffix automaton using bit-parallelism. The algorithm supports some extended patterns and other applications mentioned in Shift-Or, while keeping more efficient than Shift-Or.

8.5.6 Practical Comparison

Figure 8.20 shows a practical comparison between string matching algorithms run on our reference machine. The values are correct within 5% of accuracy with a 95% confidence interval. We tested English text from the TREC collection, DNA (corresponding to 'h.influenzae') and random text uniformly generated over 64 letters. The patterns were randomly selected from the text except for random

text, where they were randomly generated. We tested over 10 Mb of text and measured CPU time. We tested short patterns on English and random text and long patterns on DNA, which are the typical cases.

We first analyze the case of random text, where except for very short patterns the clear winners are BNDM (the bit-parallel implementation of BDM) and the BMS (Sunday) algorithm. The more classical Boyer-Moore and BDM algorithms are also very close. Among the algorithms that do not improve with the pattern length, Shift-Or is the fastest, and KMP is much slower than the naive algorithm.

The picture is similar for English text, except that we have included the Agrep software in this comparison, which worked well only on English text. Agrep turns out to be much faster than others. This is not because of using a special algorithm (it uses a BM-family algorithm) but because the code is carefully optimized. This shows the importance of careful coding as well as using good algorithms, especially in text searching where a few operations per text character are performed.

Longer patterns are shown for a DNA text. BNDM is the fastest for moderate patterns, but since it does not improve with the length after $m > w$, the classical BDM finally obtains better times. They are much better than the Boyer-Moore family because the alphabet is small and the suffix automaton technique makes better use of the information on the pattern.

We have not shown the case of extended patterns, that is, where flexibility plays a role. For this case, BNDM is normally the fastest when it can be applied (e.g., it supports classes of characters but not wild cards), otherwise Shift-Or is the best option. Shift-Or is also the best option when the text must be accessed sequentially and it is not possible to skip characters.

8.5.7 Phrases and Proximity

If a sequence of words is searched to appear in the text exactly as in the pattern (i.e., with the same separators) the problem is similar to that of exact search of a single pattern, by just forgetting the fact that there are many words. If any separator between words is to be allowed, it is possible to arrange it using an extended pattern or regular expression search.

The best way to search a phrase element-wise is to search for the element which is less frequent or can be searched faster (both criteria normally match). For instance, longer patterns are better than shorter ones; allowing fewer errors is better than allowing more errors. Once such an element is found, the neighboring words are checked to see if a complete match is found.

A similar algorithm can be used to search a proximity query.

8.6 Pattern Matching

We present in this section the main techniques to deal with complex patterns. We divide it into two main groups: searching allowing errors and searching for extended patterns.

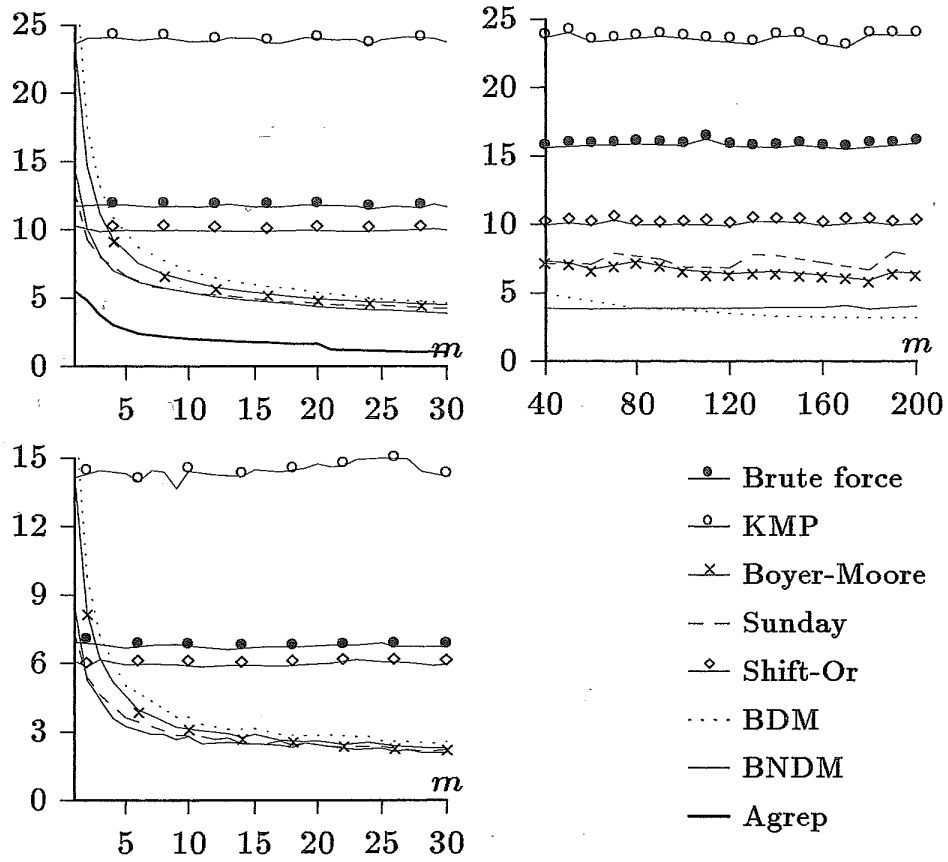


Figure 8.20 Practical comparison among algorithms. The upper left plot is for short patterns on English text. The upper right one is for long patterns on DNA. The lower plot is for short patterns on random text (on 64 letters). Times are in tenths of seconds per megabyte.

8.6.1 String Matching Allowing Errors

This problem (called 'approximate string matching') can be stated as follows: given a short pattern P of length m, a long text T of length n, and a maximum allowed number of errors k, find all the text positions where the pattern occurs with at most k errors. This statement corresponds to the Levenshtein distance. With minimal modifications it is adapted to searching whole words matching the pattern with k errors.

This problem is newer than exact string matching, although there are already a number of solutions. We sketch the main approaches.

Dynamic Programming

The classical solution to approximate string matching is based on dynamic programming. A matrix $C[0..m, 0..n]$ is filled column by column, where $C[i, j]$

represents the minimum number of errors needed to match $P_{1..i}$ to a suffix of $T_{1..j}$. This is computed as follows

$$\begin{aligned} C[0,j] &= 0 \\ C[i,0] &= i \\ C[i,j] &= \text{if } (P_i = T_j) \text{ then } C[i-1,j-1] \\ &\quad \text{else } 1 + \min(C[i-1,j], C[i,j-1], C[i-1,j-1]) \end{aligned}$$

where a match is reported at text positions j such that $C[m,j] \leq k$ (the final positions of the occurrences are reported).

Therefore, the algorithm is $O(mn)$ time. Since only the previous column of the matrix is needed, it can be implemented in $O(m)$ space. Its preprocessing time is $O(m)$. Figure 8.21 illustrates this algorithm.

In recent years several algorithms have been presented that achieve $O(kn)$ time in the worst case or even less in the average case, by taking advantage of the properties of the dynamic programming matrix (e.g., values in neighbor cells differ at most by one).

Automaton

It is interesting to note that the problem can be reduced to a non-deterministic finite automaton (NFA). Consider the NFA for $k = 2$ errors shown in Figure 8.22. Each row denotes the number of errors seen. The first one 0, the second one 1, and so on. Every column represents matching the pattern up to a given position. At each iteration, a new text character is read and the automaton changes its states. Horizontal arrows represent matching a character, vertical arrows represent insertions into the pattern, solid diagonal arrows represent replacements, and dashed diagonal arrows represent deletions in the pattern (they are ε-transitions). The automaton accepts a text position as the end of a match

		s	u	r	g	e	r	y
	0	0	0	0	0	0	0	0
s	1	0	1	1	1	1	1	1
u	2	1	0	1	2	2	2	2
r	3	2	1	0	1	2	2	3
v	4	3	2	1	1	2	3	3
e	5	4	3	2	2	1	2	3
y	6	5	4	3	3	**2**	**2**	**2**

Figure 8.21 The dynamic programming algorithm search 'survey' in the text 'surgery' with two errors. Bold entries indicate matching positions.

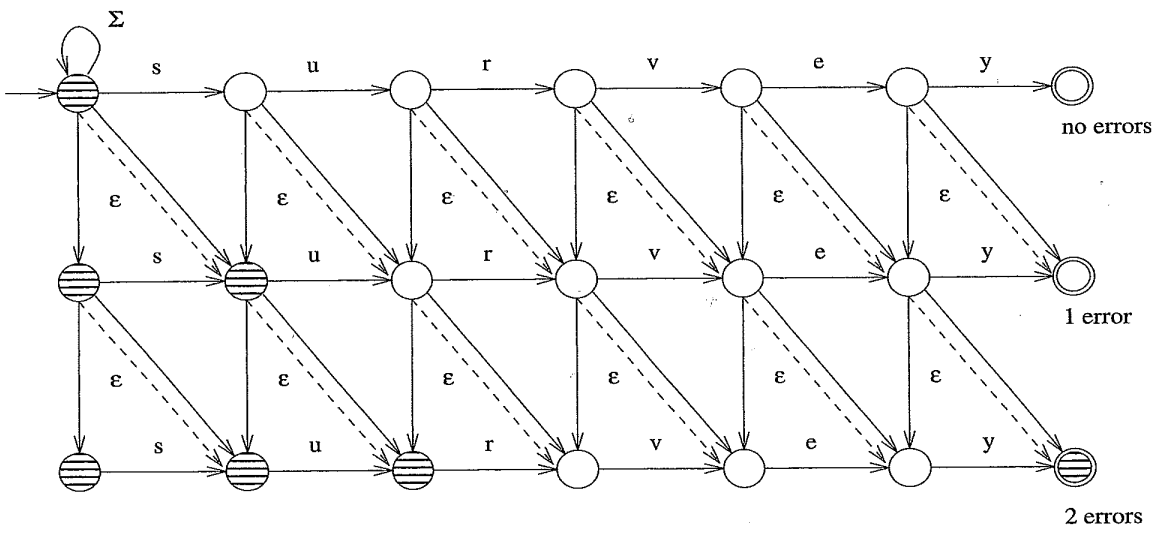


Figure 8.22 An NFA for approximate string matching of the pattern '`survey`' with two errors. The shaded states are those active after reading the text '`surgery`'. Unlabelled transitions match any character.

with k errors whenever the $(k+1)$-th rightmost state is active.

It is not hard to see that once a state in the automaton is active, all the states of the same column and higher rows are active too. Moreover, at a given text character, if we collect the smallest active rows at each column, we obtain the current column of the dynamic programming algorithm. Figure 8.22 illustrates this (compare the figure with Figure 8.21).

One solution is to make this automaton deterministic (DFA). Although the search phase is $O(n)$, the DFA can be huge. An alternative solution is based on bit-parallelism and is explained next.

Bit-Parallelism

Bit-parallelism has been used to parallelize the computation of the dynamic programming matrix (achieving average complexity $O(kn/w)$) and to parallelize the computation of the NFA (without converting it to deterministic), obtaining $O(kmn/w)$ time in the worst case. Such algorithms achieve $O(n)$ search time for short patterns and are currently the fastest ones in many cases, running at 6 to 10 Mb per second on our reference machine.

Filtering

Finally, other approaches first filter the text, reducing the area where dynamic programming needs to be used. These algorithms achieve 'sublinear' expected time in many cases for low error ratios (i.e., not all text characters are inspected,

$O(kn \log_\sigma(m)/m)$ is a typical figure), although the filtration is not effective for more errors. Filtration is based on the fact that some portions of the pattern must appear with no errors even in an approximate occurrence.

The fastest algorithm for low error levels is based on filtering: if the pattern is split into $k+1$ pieces, any approximate occurrence must contain at least one of the pieces with no errors, since k errors cannot alter all the $k+1$ pieces. Hence, the search begins with a multipattern exact search for the pieces and it later verifies the areas that may contain a match (using another algorithm).

8.6.2 Regular Expressions and Extended Patterns

General regular expressions are searched by building an automaton which finds all their occurrences in a text. This process first builds a non-deterministic finite automaton of size $O(m)$, where m is the length of the regular expression. The classical solution is to convert this automaton to deterministic form. A deterministic automaton can search any regular expression in $O(n)$ time. However, its size and construction time can be exponential in m, i.e. $O(m2^m)$. See Figure 8.23. Excluding preprocessing, this algorithm runs at 6 Mb/sec in the reference machine.

Recently the use of bit-parallelism has been proposed to avoid the construction of the deterministic automaton. The non-deterministic automaton is simulated instead. One bit per automaton state is used to represent whether the state is active or not. Due to the algorithm used to build the non-deterministic automaton, all the transitions move forward except for ε-transitions. The idea is that for each text character two steps are carried out. The first one moves forward, and the second one takes care of all the ε-transitions. A function E from bit masks to bit masks is precomputed so that all the corresponding bits are moved according to the ε-transitions. Since this function is very large (i.e., 2^m entries) its domain is split in many functions from 8- or 16-bit submasks to m-bit masks. This is possible because $E(B_1...B_j) = E(B_1)|...|E(B_j)$, where B_i

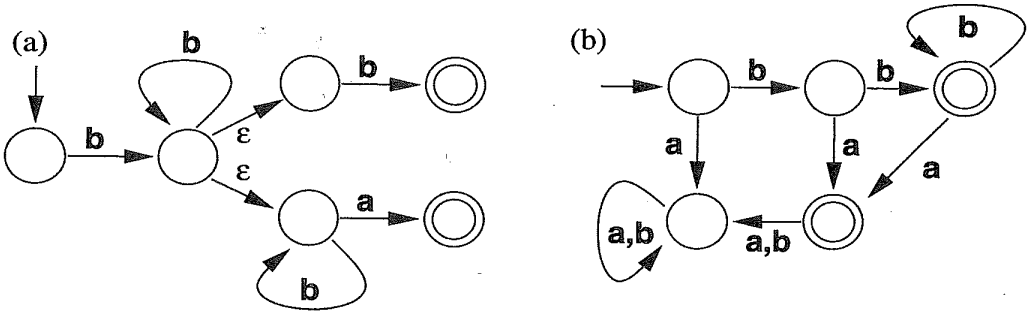


Figure 8.23 The non-deterministic (a) and deterministic (b) automata for the regular expression `b b*  (b | b*a)`.

are the submasks. Hence, the scheme performs $\lceil m/8 \rceil$ or $\lceil m/16 \rceil$ operations per text character and needs $\lceil m/8 \rceil 2^8 \lceil m/w \rceil$ or $\lceil m/16 \rceil 2^{16} \lceil m/w \rceil$ machine words of memory.

Extended patterns can be rephrased as regular expressions and solved as before. However, in many cases it is more efficient to give them a specialized solution, as we saw for the extensions of exact searching (bit-parallel algorithms). Moreover, extended patterns can be combined with approximate search for maximum flexibility. In general, the bit-parallel approach is the best equipped to deal with extended patterns.

Real times for regular expressions and extended pattern searching using this technique are between 2–8 Mb/sec.

8.6.3 Pattern Matching Using Indices

We end this section by explaining how the indexing techniques we presented for simple searching of words can in fact be extended to search for more complex patterns.

Inverted Files

As inverted files are word-oriented, other types of queries such as suffix or substring queries, searching allowing errors and regular expressions, are solved by a *sequential* (i.e., online) search over the vocabulary. This is not too bad since the size of the vocabulary is small with respect to the text size.

After either type of search, a list of vocabulary words that matched the query is obtained. All their lists of occurrences are now *merged* to retrieve a list of documents and (if required) the matching text positions. If block addressing is used and the positions are required or the blocks do not coincide with the retrieval unit, the search must be completed with a sequential search over the blocks.

Notice that an inverted index is word-oriented. Because of that it is not surprising that it is not able to efficiently find approximate matches or regular expressions that span many words. This is a restriction of this scheme. Variations that are not subject to this restriction have been proposed for languages which do not have a clear concept of word, like Finnish. They collect text *samples* or *n-grams*, which are fixed-length strings picked at regular text intervals. Searching is in general more powerful but more expensive.

In a full-inverted index, search times for simple words allowing errors on 250 Mb of text took our reference machine from 0.6 to 0.85 seconds, while very complex expressions on extended patterns took from 0.8 to 3 seconds. As a comparison, the same collection cut in blocks of 1 Mb size takes more than 8 seconds for an approximate search with one error and more than 20 for two errors.

Suffix Trees and Suffix Arrays

If the suffix tree indexes all text positions it can search for words, prefixes, suffixes and substrings with the same search algorithm and cost described for word search. However, indexing all positions makes the index 10 to 20 times the text size for suffix trees.

Range queries are easily solved too, by just searching both extremes in the trie and then collecting all the leaves which lie in the middle. In this case the cost is the height of the tree, which is $O(\log n)$ on average (excluding the tasks of collecting and sorting the leaves).

Regular expressions can be searched in the suffix tree. The algorithm simply simulates sequential searching of the regular expression. It begins at the root, since any possible match starts there too. For each child of the current node labeled by the character c, it assumes that the next text character is c and recursively enters into that subtree. This is done for each of the children of the current node. The search stops only when the automaton has no transition to follow. It has been shown that for random text only $O(n^{\alpha}\text{polylog}(n))$ nodes are traversed (for $0 < \alpha < 1$ dependent on the regular expression). Hence, the search time is sublinear for regular expressions *without* the restriction that they must occur inside a word. Extended patterns can be searched in the same way by taking them as regular expressions.

Unrestricted approximate string matching is also possible using the same idea. We present a simplified version here. Imagine that the search is online and traverse the tree recursively as before. Since all suffixes start at the root, any match starts at the root too, and therefore do not allow the match to start later. The search will automatically stop at depth $m + k$ at most (since at that point more than k errors have occurred). This implies constant search time if n is large enough (albeit exponential on m and k). Other problems such as approximate search of extended patterns can be solved in the same way, using the appropriate online algorithm.

Suffix trees are able to perform other complex searches that we have not considered in our query language (see Chapter 4). These are specialized operations which are useful in specific areas. Some examples are: find the longest substring in the text that appears more than once, find the most common substring of a fixed size, etc.

If a suffix array indexes all text positions, *any* algorithm that works on suffix trees at $C(n)$ cost will work on suffix arrays at $O(C(n) \log n)$ cost. This is because the operations performed on the suffix tree consist of descending to a child node, which is done in $O(1)$ time. This operation can be simulated in the suffix array in $O(\log n)$ time by binary searching the new boundaries (each suffix tree node corresponds to a string, which can be mapped to the suffix array interval holding all suffixes starting with that string). Some patterns can be searched directly in the suffix array in $O(\log n)$ total search time without simulating the suffix tree. These are: word, prefix, suffix and subword search, as well as range search.

However, again, indexing all text positions normally makes the suffix array

size four times or more the text size. A different alternative for suffix arrays is to index only word beginnings and to use a vocabulary supra-index, using the same search algorithms used for the inverted lists.

8.7 Structural Queries

The algorithms to search on structured text (see Chapter 4) are largely dependent on each model. We extract their common features in this section.

A first concern about this problem is how to store the structural information. Some implementations build an ad hoc index to store the structure. This is potentially more efficient and independent of any consideration about the text. However, it requires extra development and maintenance effort.

Other techniques assume that the structure is marked in the text using 'tags' (i.e., strings that identify the structural elements). This is the case with HTML text but not the case with C code where the marks are implicit and are inherent to C. The technique relies on the same index to query content (such as inverted files), using it to index and search those tags as if they were words. In many cases this is as efficient as an ad hoc index, and its integration into an existing text database is simpler. Moreover, it is possible to define the structure dynamically, since the appropriate tags can be selected at search time. For that goal, inverted files are better since they naturally deliver the results in text order, which makes the structure information easier to obtain. On the other hand, some queries such as direct ancestry are hard to answer without an ad hoc index.

Once the content and structural elements have been found by using some index, a set of answers is generated. The models allow further operations to be applied on those answers, such as 'select all areas in the left-hand argument which contain an area of the right-hand argument.' This is in general solved in a way very similar to the set manipulation techniques already explained in section 8.4. However, the operations tend to be more complex, and it is not always possible to find an evaluation algorithm which has linear time with respect to the size of the intermediate results.

It is worth mentioning that some models use completely different algorithms, such as exhaustive search techniques for tree pattern matching. Those problems are NP-complete in many cases.

8.8 Compression

In this section we discuss the issues of searching compressed text directly and of searching compressed indices. Compression is important when available storage is a limiting factor, as is the case of indexing the Web.

Searching and compression were traditionally regarded as exclusive operations. Texts which were not to be searched could be compressed, and to search

a compressed text it had to be decompressed first. In recent years, very efficient compression techniques have appeared that allow searching directly in the compressed text. Moreover, the search performance is *improved*, since the CPU times are similar but the disk times are largely reduced. This leads to a win-win situation.

Discussion on how common text and lists of numbers can be compressed has been covered in Chapter 7.

8.8.1 Sequential Searching

A few approaches to directly searching compressed text exist. One of the most successful techniques in practice relies on Huffman coding taking words as symbols. That is, consider each different text word as a symbol, count their frequencies, and generate a Huffman codefor the words. Then, compress the text by replacing each word with its code. To improve compression/decompression efficiency, the Huffman code uses an alphabet of bytes instead of bits. This scheme compresses faster and better than known commercial systems, even those based on Ziv-Lempel coding.

Since Huffman coding needs to store the codes of each symbol, this scheme has to store the whole vocabulary of the text, i.e. the list of all different text words. This is fully exploited to efficiently search complex queries. Although according to Heaps' law the vocabulary (i.e., the alphabet) grows as $O(n^\beta)$ for $0 < \beta < 1$, the generalized Zipf's law shows that the distribution is skewed enough so that the entropy remains constant (i.e., the compression ratio will not degrade as the text grows). Those laws are explained in Chapter 6.

Any single-word or pattern query is first searched in the vocabulary. Some queries can be binary searched, while others such as approximate searching or regular expression searching must traverse sequentially all the vocabulary. This vocabulary is rather small compared to the text size, thanks to Heaps' law. Notice that this process is exactly the same as the vocabulary searching performed by inverted indices, either for simple or complex pattern matching.

Once that search is complete, the list of different words that match the query is obtained. The Huffman codes of all those words are collected and they are searched in the compressed text. One alternative is to traverse byte-wise the compressed text and traverse the Huffman decoding tree in synchronization, so that each time that a leaf is reached, it is checked whether the leaf (i.e., word) was marked as 'matching' the query or not. This is illustrated in Figure 8.24. Boyer-Moore filtering can be used to speed up the search.

Solving phrases is a little more difficult. Each element is searched in the vocabulary. For each word of the vocabulary we define a bit mask. We set the i-th bit in the mask of all words which match with the i-th element of the phrase query. This is used together with the Shift-Or algorithm. The text is traversed byte-wise, and only when a leaf is reached, does the Shift-Or algorithm consider that a new text symbol has been read, whose bit mask is that of the leaf (see Figure 8.24). This algorithm is surprisingly simple and efficient.

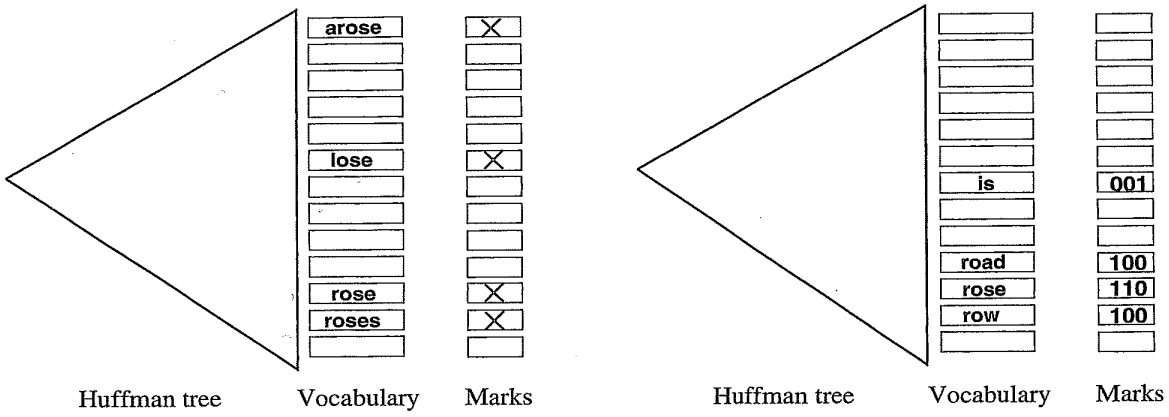


Figure 8.24 On the left, searching for the simple pattern '`rose`' allowing one error. On the right, searching for the phrase '`ro* rose is`,' where '`ro*`' represents a prefix search.

This scheme is especially fast when it comes to solving a complex query (regular expression, extended pattern, approximate search, etc.) that would be slow with a normal algorithm. This is because the complex search is done only in the small vocabulary, after which the algorithm is largely insensitive to the complexity of the originating query. Its CPU times for a simple pattern are slightly higher than those of Agrep (briefly described in section 8.5.6). However, if the I/O times are considered, compressed searching is faster than all the online algorithms. For complex queries, this scheme is unbeaten by far.

On the reference machine, the CPU times are 14 Mb/sec for any query, while for simple queries this improves to 18 Mb/sec if the speedup technique is used. Agrep, on the other hand, runs at 15 Mb/sec on simple searches and at 1–4 Mb/sec for complex ones. Moreover, I/O times are reduced to one third on the compressed text.

8.8.2 Compressed Indices

Inverted Files

Inverted files are quite amenable to compression. This is because the lists of occurrences are in increasing order of text position. Therefore, an obvious choice is to represent the differences between the previous position and the current one. These differences can be represented using less space by using techniques that favor small numbers (see Chapter 7). Notice that, the longer the lists, the smaller the differences. Reductions in 90% for block-addressing indices with blocks of 1 Kb size have been reported.

It is important to notice that compression does not necessarily degrade time performance. Most of the time spent in answering a query is in the disk transfer. Keeping the index compressed allows the transference of less data, and it may be worth the CPU work of decompressing. Notice also that the lists of

occurrences are normally traversed in a sequential manner, which is not affected by a differential compression. Query times on compressed or decompressed indices are reported to be roughly similar.

The text can also be compressed independently of the index. The text will be decompressed only to display it, or to traverse it in case of block addressing. Notice in particular that the online search technique described for compressed text in section 8.8.1 uses a vocabulary. It is possible to integrate both techniques (compression and indexing) such that they share the same vocabulary for both tasks and they do not decompress the text to index or to search.

Suffix Trees and Suffix Arrays

Some efforts to compress suffix trees have been pursued. Important reductions of the space requirements have been obtained at the cost of more expensive searching. However, the reduced space requirements happen to be similar to those of uncompressed suffix arrays, which impose much smaller performance penalties.

Suffix arrays are very hard to compress further. This is because they represent an almost perfectly random permutation of the pointers to the text.

However, the subject of building suffix arrays on compressed text has been pursued. Apart from reduced space requirements (the index plus the compressed text take less space than the uncompressed text), the main advantage is that both index construction and querying almost double their performance. Construction is faster because more compressed text fits in the same memory space, and therefore fewer text blocks are needed. Searching is faster because a large part of the search time is spent in disk seek operations over the text area to compare suffixes. If the text is smaller, the seeks reduce proportionally.

A compression technique very similar to that shown in section 8.8.1 is used. However, the Huffman code on words is replaced by a Hu-Tucker coding. The Hu-Tucker code respects the lexicographical relationships between the words, and therefore direct binary search over the compressed text is possible (this is necessary at construction and search time). This code is suboptimal by a very small percentage (2–3% in practice, with an analytical upper bound of 5%).

Indexing times for 250 Mb of text on the reference machine are close to 1.6 Mb/min if compression is used, while query times are reduced to 0.5 seconds in total and 0.3 seconds for the text alone. Supra-indices should reduce the total search time to 0.15 seconds.

Signature Files

There are many alternative ways to compress signature files. All of them are based on the fact that only a few bits are set in the whole file. It is then possible

to use efficient methods to code the bits which are not set, for instance run-length encoding. Different considerations arise if the file is stored as a sequence of bit masks or with one file per bit of the mask. They allow us to reduce space and hence disk times, or alternatively to increase B (so as to reduce the false drop probability) keeping the same space overhead. Compression ratios near 70% are reported.

8.9 Trends and Research Issues

In this chapter we covered extensively the current techniques of dealing with text retrieval. We first covered indices and then online searching. We then reviewed set manipulation, complex pattern matching and finally considered compression techniques. Figure 8.25 summarizes the tradeoff between the space needed for the index and the time to search one single word.

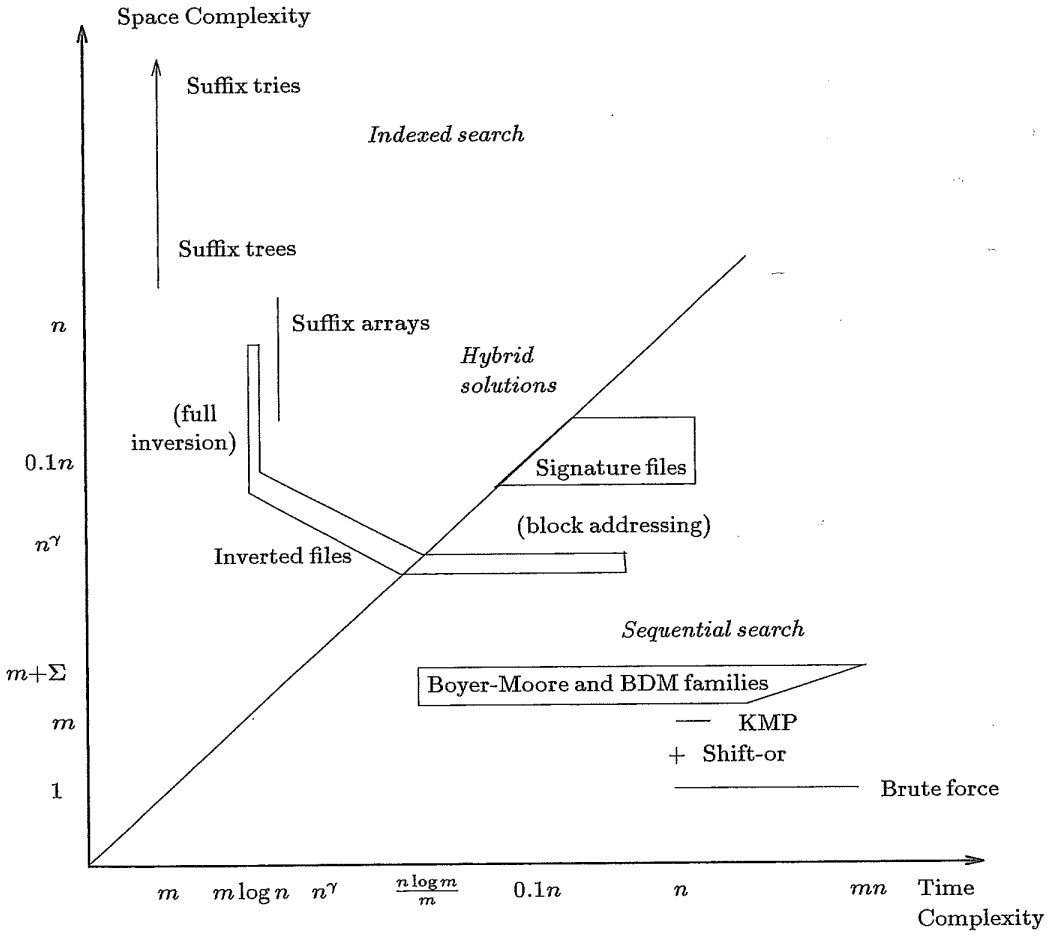


Figure 8.25 Tradeoff of index space versus word searching time.

Probably the most adequate indexing technique in practice is the inverted file. As we have shown throughout the chapter, many hidden details in other structures make them harder to use and less efficient in practice, as well as less flexible for dealing with new types of queries. These structures, however, still find application in restricted areas such as genetic databases (for suffix trees and arrays, for the relatively small texts used and their need to pose specialized queries) or some office systems (for signature files, because the text is rarely queried in fact).

The main trends in indexing and searching textual databases today are

- **Text collections are becoming huge.** This poses more demanding requirements at all levels, and solutions previously affordable are not any more. On the other hand, the speed of the processors and the relative slowness of external devices have changed what a few years ago were reasonable options (e.g., it is better to keep a text compressed because reading less text from disk and decompressing in main memory pays off).
- **Searching is becoming more complex.** As the text databases grow and become more heterogeneous and error-prone, enhanced query facilities are required, such as exploiting the text structure or allowing errors in the text. Good support for extended queries is becoming important in the evaluation of a text retrieval system.
- **Compression is becoming a star in the field.** Because of the changes mentioned in the time cost of processors and external devices, and because of new developments in the area, text retrieval and compression are no longer regarded as disjoint activities. Direct indexing and searching on compressed text provides better (sometimes much better) time performance and less space overhead at the same time. Other techniques such as block addressing trade space for processor time.

8.10 Bibliographic Discussion

A detailed explanation of a full inverted index and its construction and querying process can be found in [26]. This work also includes an analysis of the algorithms on inverted lists using the distribution of natural language. The in-place construction is described in [572]. Another construction algorithm is presented in [341].

The idea of block addressing inverted indices was first presented in a system called Glimpse [540], which also first exposed the idea of performing complex pattern matching using the vocabulary of the inverted index. Block addressing indices are analyzed in [42], where some performance improvements are proposed. The variant that indexes sequences instead of words has been implemented in a system called Grampse, which is described in [497].

Suffix arrays were presented in [538] together with the algorithm to build them in $O(n \log n)$ character comparisons. They were independently discovered

by [309] under the name of 'PAT arrays.' The algorithm to build large suffix arrays is presented in [311]. The use of supra-indices over suffix array is proposed in [37], while the modified binary search techniques to reduce disk seek time are presented in [56]. The linear-time construction of suffix trees is described in [780].

The material on signature files is based on [243]. The different alternative ways of storing the signature file are explained in [242].

The original references for the sequential search algorithms are: KMP [447], BM [110], BMH [376], BMS [751], Shift-Or [39], BDM [205] and BNDM [592]. The multipattern versions are found in [9, 179], and MultiBDM in [196]. Many enhancements of bit-parallelism to support extended patterns and allow errors are presented in [837]. Many ideas from that paper were implemented in a widely distributed software for online searching called Agrep [836].

The reader interested in more details about sequential searching algorithms may look for the original references or in good books on algorithms such as [310, 196].

One source for the classical solution to approximate string matching is [716]. An $O(kn)$ worst-case algorithm is described in [480]. The use of a DFA is proposed in [781]. The bit-parallel approach to this problem started in [837], although currently the fastest bit-parallel algorithms are [583] and [43]. Among all the filtering algorithms, the fastest one in practice is based on an idea presented in [837], later enhanced in [45], and finally implemented in [43].

A good source from which to learn about regular expressions and building a DFA is [375]. The bit-parallel implementation of the NFA is explained in [837]. Regular expression searching on suffix trees is described in [40], while searching allowing errors is presented in [779].

The Huffman coding was first presented in [386], while the word-oriented alternative is proposed in [571]. Sequential searching on text compressed using that technique is described in [577]. Compression used in combination with inverted files is described in [850], with suffix trees in [430], with suffix arrays in [575], and with signature files in [243, 242]. A good general reference on compression is [78].

Chapter 10
User Interfaces and Visualization

by Marti A. Hearst

10.1 Introduction

This chapter discusses user interfaces for communication between human information seekers and information retrieval systems. Information seeking is an imprecise process. When users approach an information access system they often have only a fuzzy understanding of how they can achieve their goals. Thus the user interface should aid in the understanding and expression of information needs. It should also help users formulate their queries, select among available information sources, understand search results, and keep track of the progress of their search.

The human-computer interface is less well understood than other aspects of information retrieval, in part because humans are more complex than computer systems, and their motivations and behaviors are more difficult to measure and characterize. The area is also undergoing rapid change, and so the discussion in this chapter will emphasize recent developments rather than established wisdom.

The chapter will first outline the human side of the information seeking process and then focus on the aspects of this process that can best be supported by the user interface. Discussion will encompass current practice and technology, recently proposed innovative ideas, and suggestions for future areas of development.

Section 10.2 outlines design principles for human-computer interaction and introduces notions related to information visualization. section 10.3 describes information seeking models, past and present. The next four sections describe user interface support for starting the search process, for query specification, for viewing retrieval results in context, and for interactive relevance feedback. The last major section, section 10.8, describes user interface techniques to support the information access process as a whole. Section 10.9 speculates on future developments and Section 10.10 provides suggestions for further reading. Figure 10.1 presents the flow of the chapter contents.

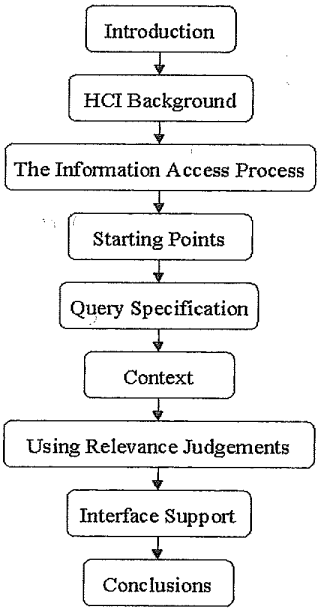


Figure 10.1 The flow of this chapter's contents.

10.2 Human-Computer Interaction

What makes an effective human-computer interface? Ben Shneiderman, an expert in the field, writes [725, p.10]:

> Well designed, effective computer systems generate positive feelings of success, competence, mastery, and clarity in the user community. When an interactive system is well-designed, the interface almost disappears, enabling users to concentrate on their work, exploration, or pleasure.

As steps towards achieving these goals, Shneiderman lists principles for design of user interfaces. Those which are particularly important for information access include (slightly restated): provide informative feedback, permit easy reversal of actions, support an internal locus of control, reduce working memory load, and provide alternative interfaces for novice and expert users. Each of these principles should be instantiated differently depending on the particular interface application. Below we discuss those principles that are of special interest to information access systems.

10.2.1 Design Principles

Offer informative feedback. This principle is especially important for information access interfaces. In this chapter we will see current ideas about how to provide

users with feedback about the relationship between their query specification and documents retrieved, about relationships among retrieved documents, and about relationships between retrieved documents and metadata describing collections. If the user has control of how and when feedback is provided, then the system provides an *internal locus of control.*

Reduce working memory load. Information access is an iterative process, the goals of which shift and change as information is encountered. One key way information access interfaces can help with memory load is to provide mechanisms for keeping track of choices made during the search process, allowing users to return to temporarily abandoned strategies, jump from one strategy to the next, and retain information and context across search sessions. Another memory-aiding device is to provide browsable information that is relevant to the current stage of the information access process. This includes suggestions of related terms or metadata, and search starting points including lists of sources and topic lists.

Provide alternative interfaces for novice and expert users. An important tradeoff in all user interface design is that of simplicity versus power. Simple interfaces are easier to learn, at the expense of less flexibility and sometimes less efficient use. Powerful interfaces allow a knowledgeable user to do more and have more control over the operation of the interface, but can be time-consuming to learn and impose a memory burden on people who use the system only intermittently. A common solution is to use a 'scaffolding' technique [684]. The novice user is presented with a simple interface that can be learned quickly and that provides the basic functionality of the application, but is restricted in power and flexibility. Alternative interfaces are offered for more experienced users, giving them more control, more options, and more features, or potentially even entirely different interaction models. Good user interface design provides intuitive bridges between the simple and the advanced interfaces.

Information access interfaces must contend with special kinds of simplicity/power tradeoffs. One such tradeoff is the amount of information shown about the workings of the search system itself. Users who are new to a system or to a particular collection may not know enough about the system or the domain associated with the collection to make choices among complex features. They may not know how best to weight terms, or in the case of relevance feedback, not know what the effects of reweighting terms would be. On the other hand, users that have worked with a system and gotten a feeling for a topic are likely to be able to choose among suggested terms to add to their query in an informed manner. Determining how much information to show the user of the system is a major design choice in information access interfaces.

10.2.2 The Role of Visualization

The tools of computer interface design are familiar to most computer users today: windows, menus, icons, dialog boxes, and so on. These make use of bit-mapped display and computer graphics to provide a more accessible interface

than command-line-based displays. A less familiar but growing area is that of *information visualization*, which attempts to provide visual depictions of very large information spaces.

Humans are highly attuned to images and visual information [769, 456, 483]. Pictures and graphics can be captivating and appealing, especially if well designed. A visual representation can communicate some kinds of information much more rapidly and effectively than any other method. Consider the difference between a written description of a person's face and a photograph of it, or the difference between a table of numbers containing a correlation and a scatter plot showing the same information.

The growing prevalence of fast graphics processors and high resolution color monitors is increasing interest in information visualization. Scientific visualization, a rapidly advancing branch of this field, maps physical phenomena onto two- or three-dimensional representations [433]. An example of scientific visualization is a colorful image of the pattern of peaks and valleys on the ocean floor; this provides a view of physical phenomena for which a photograph cannot (currently) be taken. Instead, the image is constructed from data that represent the underlying phenomena.

Visualization of inherently abstract information is more difficult, and visualization of textually represented information is especially challenging. Language is our main means of communicating abstract ideas for which there is no obvious physical manifestation. What does a picture look like that describes negotiations over a trade agreement in which one party demands concessions on environmental policies while the other requires help in strengthening its currency?

Despite the difficulties, researchers are attempting to represent aspects of the information access process using information visualization techniques. Some of these will be described later in this chapter. Aside from using *icons* and *color highlighting*, the main information visualization techniques include *brushing and linking* [233, 773], *panning and zooming* [71], *focus-plus-context* [502], *magic lenses* [95], and the use of *animation* to retain context and help make occluded information visible [676, 143]. These techniques support dynamic, interactive use. Interactivity seems to be an especially important property for visualizing abstract information, although it has not played as large a role within scientific visualization.

Brushing and linking refers to the connecting of two or more views of the same data, such that a change to the representation in one view affects the representation in the other views as well. For example, say a display consists of two parts: a histogram and a list of titles. The histogram shows, for a set of documents, how many documents were published each year. The title list shows the titles for the corresponding documents. Brushing and linking would allow the user to assign a color, say red, to one bar of the histogram, thus causing the titles in the list display that were published during the corresponding year to also be highlighted in red.

Panning and zooming refers to the actions of a movie camera that can scan sideways across a scene (panning) or move in for a closeup or back away to get a wider view (zooming). For example, text clustering can be used to show a

top-level view of the main themes in a document collection (see Figures 10.7 and 10.8). Zooming can be used to move 'closer,' showing individual documents as icons, and then zoom in closer still to see the text associated with an individual document.

When zooming is used, the more detail that is visible about a particular item, the less can be seen about the surrounding items. Focus-plus-context is used to partly alleviate this effect. The idea is to make one portion of the view — the focus of attention — larger, while simultaneously shrinking the surrounding objects. The farther an object is from the focus of attention, the smaller it is made to appear, like the effect seen in a fisheye camera lens (also in some door peepholes).

Magic lenses are directly manipulable transparent windows that, when overlapped on some other data type, cause a transformation to be applied to the underlying data, thus changing its appearance (see Figure 10.13). The most straightforward application of magic lenses is for drawing tasks, and it is especially useful if used as a two-handed interface. For example, the left hand can be used to position a color lens over a drawing of an object. The right hand is used to mouse-click on the lens, thus causing the appearance of the underlying object to be transformed to the color specified by the lens.

Additionally, there are a large number of graphical methods for depicting trees and hierarchies, some of which make use of animation to show nodes that would otherwise be occluded (hidden from view by other nodes) [286, 364, 407, 478, 676].

It is often useful to combine these techniques into an interface layout consisting of an *overview plus details* [321, 644]. An overview, such as a table-of-contents of a large manual, is shown in one window. A mouse-click on the title of the chapter causes the text of the chapter itself to appear in another window, in a linking action (see Figure 10.19). Panning and zooming or focus-plus-context can be used to change the view of the contents within the overview window.

10.2.3 Evaluating Interactive Systems

From the viewpoint of user interface design, people have widely differing abilities, preferences, and predilections. Important differences for information access interfaces include relative spatial ability and memory, reasoning abilities, verbal aptitude, and (potentially) personality differences [227, 725]. Age and cultural differences can contribute to acceptance or rejection of interface techniques [557]. An interface innovation can be useful and pleasing for some users, and foreign and cumbersome for others. Thus software design should allow for flexibility in interaction style, and new features should not be expected to be equally helpful for all users.

An important aspect of human-computer interaction is the methodology for evaluation of user interface techniques. Precision and recall measures have been widely used for comparing the ranking results of non-interactive systems, but are less appropriate for assessing interactive systems [470]. The standard evaluations

emphasize high recall levels; in the TREC tasks systems are compared to see how well they return the top 1000 documents (see chapter 3). However, in many interactive settings, users require only a few relevant documents and do not care about high recall to evaluate highly interactive information access systems, useful metrics beyond precision and recall include: time required to learn the system, time required to achieve goals on benchmark tasks, error rates, and retention of the use of the interface over time. Throughout this chapter, empirical results of user studies are presented whenever they are available.

Empirical data involving human users is time consuming to gather and difficult to draw conclusions from. This is due in part to variation in users' characteristics and motivations, and in part to the broad scope of information access activities. Formal psychological studies usually only uncover narrow conclusions within restricted contexts. For example, quantities such as the length of time it takes for a user to select an item from a fixed menu under various conditions have been characterized empirically [142], but variations in interaction behavior for complex tasks like information access are difficult to account for accurately. Nielsen [605] advocates a more informal evaluation approach (called heuristic evaluation) in which user interface affordances are assessed in terms of more general properties and without concern about statistically significant results.

10.3 The Information Access Process

A person engaged in an information seeking process has one or more *goals* in mind and uses a search system as a tool to help achieve those goals. Goals requiring information access can range quite widely, from finding a plumber to keeping informed about a business competitor, from writing a publishable scholarly article to investigating an allegation of fraud.

Information access *tasks* are used to achieve these goals. These tasks span the spectrum from asking specific questions to exhaustively researching a topic. Other tasks fall between these two extremes. A study of business analysts [614] found three main kinds of information seeking tasks: monitoring a well known topic over time (such as researching competitors' activities each quarter), following a plan or stereotyped series of searches to achieve a particular goal (such as keeping up to date on good business practices), and exploring a topic in an undirected fashion (as when getting to know an unfamiliar industry). Although the goals differ, there is a common core revolving around the information seeking component, which is our focus here.

10.3.1 Models of Interaction

Most accounts of the information access process assume an interaction cycle consisting of query specification, receipt and examination of retrieval results, and then either stopping or reformulating the query and repeating the process

until a perfect result set is found [700, 726]. In more detail, the standard process can be described according to the following sequence of steps (see Figure 10.2):

(1) Start with an information need.
(2) Select a system and collections to search on.
(3) Formulate a query.
(4) Send the query to the system.
(5) Receive the results in the form of information items.
(6) Scan, evaluate, and interpret the results.
(7) Either stop, or,
(8) Reformulate the query and go to step 4.

This simple interaction model (used by Web search engines) is the only model that most information seekers see today. This model does not take into account the fact that many users dislike being confronted with a long disorganized list of retrieval results that do not directly address their information needs. It also contains an underlying assumption that the user's information need is static and the information seeking process is one of successively refining a query until it retrieves all and only those documents relevant to the original information need.

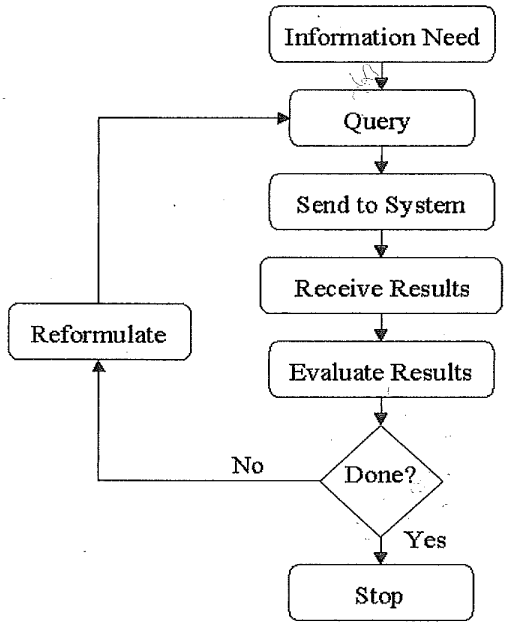


Figure 10.2 A simplified diagram of the standard model of the information access processes.

In actuality, users learn during the search process. They scan information, read the titles in result sets, read the retrieved documents themselves, viewing lists of topics related to their query terms, and navigating within hyperlinked Web sites. The recent advent of hyperlinks as a pivotal part of the information seeking process makes it no longer feasible to ignore the role of scanning and navigation within the search process itself. In particular, today a near-miss is much more acceptable than it was with bibliographic search, since an information seeker using the Web can navigate hyperlinks from a near-miss in the hopes that a useful page will be a few links away.

The standard model also downplays the interaction that takes place when the user scans terms suggested as a result of relevance feedback, scans thesaurus structures, or views thematic overviews of document collections. It deemphasizes the role of source selection, which is increasingly important now that, for the first time, tens of thousands of information collections are immediately reachable for millions of people.

Thus, while useful for describing the basics of information access systems, this simple interaction model is being challenged on many fronts [65, 614, 105, 365, 192]. Bates [65] proposes the 'berry-picking' model of information seeking, which has two main points. The first is that, as a result of reading and learning from the information encountered throughout the search process, the users' information needs, and consequently their queries, continually shift. Information encountered at one point in a search may lead in a new, unanticipated direction. The original goal may become partly fulfilled, thus lowering the priority of one goal in favor of another. This is posed in contrast to the assumption of 'standard' information retrieval that the user's information need remains the same throughout the search process. The second point is that users' information needs are not satisfied by a single, final retrieved set of documents, but rather by a series of selections and bits of information found along the way. This is in contrast to the assumption that the main goal of the search process is to hone down the set of retrieved documents into a perfect match of the original information need.

The berry-picking model is supported by a number of observational studies [236, 105], including that of O'Day and Jeffries [614]. They found that the information seeking process consisted of a series of interconnected but diverse searches on one problem-based theme. They also found that search results for a goal tended to trigger new goals, and hence search in new directions, but that the context of the problem and the previous searches was carried from one stage of search to the next. They also found that the main value of the search resided in the accumulated learning and acquisition of information that occurred during the search process, rather than in the final results set.

Thus, a user interface for information access should allow users to reassess their goals and adjust their search strategy accordingly. A related situation occurs when users encounter a 'trigger' that causes them to pursue a different strategy temporarily, perhaps to return to the current unfinished activity at a later time. An implication of these observations is that the user interface should support search strategies by making it easy to follow trails with unanticipated results. This can be accomplished in part by supplying ways to record the progress

of the current strategy and to store, find, and reload intermediate results, and by supporting pursuit of multiple strategies simultaneously.

The user interface should also support methods for monitoring the status of the current strategy in relation to the user's current task and high-level goals. One way to cast the activity of monitoring the progress of a search strategy relative to a goal or subgoal is in terms of a cost/benefit analysis, or an analysis of diminishing returns [690]. This kind of analysis assumes that at any point in the search process, the user is pursuing the strategy that has the highest expected utility. If, as a consequence of some local tactical choices, another strategy presents itself as being of higher utility than the current one, the current one is (temporarily or permanently) abandoned in favor of the new strategy.

There are a number of theories and frameworks that contrast *browsing*, *querying*, *navigating*, and *scanning* along several dimensions [75, 159, 542, 804]. Here we assume that users scan information structure, be it titles, thesaurus terms, hyperlinks, category labels, or the results of clustering, and then either select a displayed item for some purpose (to read in detail, to use as input to a query, to navigate to a new page of information) or formulate a query (either by recalling potential words or by selecting categories or suggested terms that have been scanned). In both cases, a new set of information is then made viewable for scanning. Queries tend to produce new, ad hoc collections of information that have not been gathered together before, whereas selection retrieves information that has already been composed or organized. Navigation refers to following a chain of links, switching from one view to another, toward some goal, in a sequence of scan and select operations. Browsing refers to the casual, mainly undirected exploration of information structures, and is usually done in tandem with selection, although queries can also be used to create subcollections to browse through. An important aspect of the interaction process is that the output of one action should be easily used as the input to the next.

10.3.2 Non-Search Parts of the Information Access Process

The O'Day and Jeffries study [614] found that information seeking is only one part of the full work process their subjects were engaged in. In between searching sessions many different kinds of work was done with the retrieved information, including reading and annotating [617] and analysis. O'Day and Jeffries examined the analysis steps in more detail, finding that 80% of this work fell into six main types: finding trends, making comparisons, aggregating information, identifying a critical subset, assessing, and interpreting. The remaining 20% consisted of cross-referencing, summarizing, finding evocative visualizations for reports, and miscellaneous activities. The Sensemaking work of Russell *et al.* [690] also discusses information work as a process in which information retrieval plays only a small part. They observe that most of the effort made in Sensemaking is in the synthesis of a good representation, or ways of thinking about, the problem at hand. They describe the process of formulating and crystallizing the important concepts for a given task.

From these observations it is convenient to divide the entire information access process into two main components: search/retrieval, and analysis/synthesis of results. User interfaces should allow both kinds of activity to be tightly interwoven. However, analysis/synthesis are activities that can be done independently of information seeking, and for our purposes it is useful to make a distinction between the two types of activities.

10.3.3 Earlier Interface Studies

The bulk of the literature on studies of human-computer information seeking behavior concerns information intermediaries using online systems consisting of bibliographic records (e.g., [546, 707, 104]), sometimes with costs assessed per time unit. Unfortunately, many of the assumptions behind those studies do not reflect the conditions of modern information access [335, 222]. The differences include the following:

- The text being searched now is often full text rather than bibliographic citations. Because users have access to full text, rather than document surrogates, it is more likely that simple queries will find relevant answers directly as part of the search process.
- Modern systems use statistical ranking (which is more effective when abstracts and full text are available than when only titles and citations are available) whereas most studies were performed on Boolean systems.
- Much of modern searching is done by end users, many new to online searching, rather than professional intermediaries, which were the focus of many of the earlier studies.
- Tens of thousands of sources are now available online on networked information systems, and many are tightly coupled via hyperlinks, as opposed to being stored in separate collections owned by separate services. Earlier studies generally used systems in which moving from one collection to another required prior knowledge of the collections and considerable time and effort to switch. A near miss is much more useful in this hyperlinked environment than in earlier systems, since hyperlinks allow users to navigate from the near miss directly to the source containing information of interest. In a card catalog environment, where documents are represented as isolated units, a near miss consists of finding a book in the general area of interest and then going to the bookshelf in the library to look for related books, or obtaining copies of many issues of a journal and scanning for related articles.
- Finally, most users have access to bit-mapped displays allowing for direct manipulation, or at least form fillin. Most earlier studies and bibliographic systems were implemented on TTY displays, which require command-line based syntax and do a poor job of retaining context.

Despite these significant differences, some general information seeking strategies have been identified that seem to transfer across systems. Additionally, although modern systems have remedied many of the problems of earlier online public access catalogs, they also introduce new problems of their own.

10.4 Starting Points

Search interfaces must provide users with good ways to get started. An empty screen or a blank entry form does not provide clues to help a user decide how to start the search process. Users usually do not begin by creating a long, detailed expression of their information need. Studies show that users tend to start out with very short queries, inspect the results, and then modify those queries in an incremental feedback cycle [22]. The initial query can be seen as a kind of 'testing the water' to see what kinds of results are returned and get an idea of how to reformulate the query [804, 65]. Thus, one task of an information access interface is to help users select the sources and collections to search on.

For example, there are many different information sources associated with cancer, and there are many different kinds of information a user might like to know about cancer. Guiding the user to the right set of starting points can help with the initial problem formulation. Traditional bibliographic search assumes that the user begins by looking through a list of names of sources and choosing which collections to search on, while Web search engines obliterate the distinctions between sources and plunge the user into the middle of a Web site with little information about the relationship of the search hit to the rest of the collection. In neither case is the interface to the available sources particularly helpful.

In this section we will discuss four main types of starting points: *lists*, *overviews*, *examples*, and *automated source selection*.

10.4.1 Lists of Collections

Typical online systems such as LEXIS-NEXIS require users to begin any inquiry with a scan through a long list of source names and guess which ones will be of interest. Usually little information beyond the name of the collection is provided online for these sources (see Figure 10.3). If the user is not satisfied with the results on one collection, they must reissue the query on another collection.

Frequent searchers eventually learn a set of sources that are useful for their domains of interest, either through experience, formal training, or recommendations from friends and colleagues. Often-used sources can be stored on a 'favorites' list, also known as a *bookmark* list or a *hotlist* on the Web. Recent research explores the maintenance of a personalized information profile for users or work groups, based on the kinds of information they've used in the past [277].

However, when users want to search outside their domains of expertise, a list of familiar sources is not sufficient. Professional searchers such as librarians

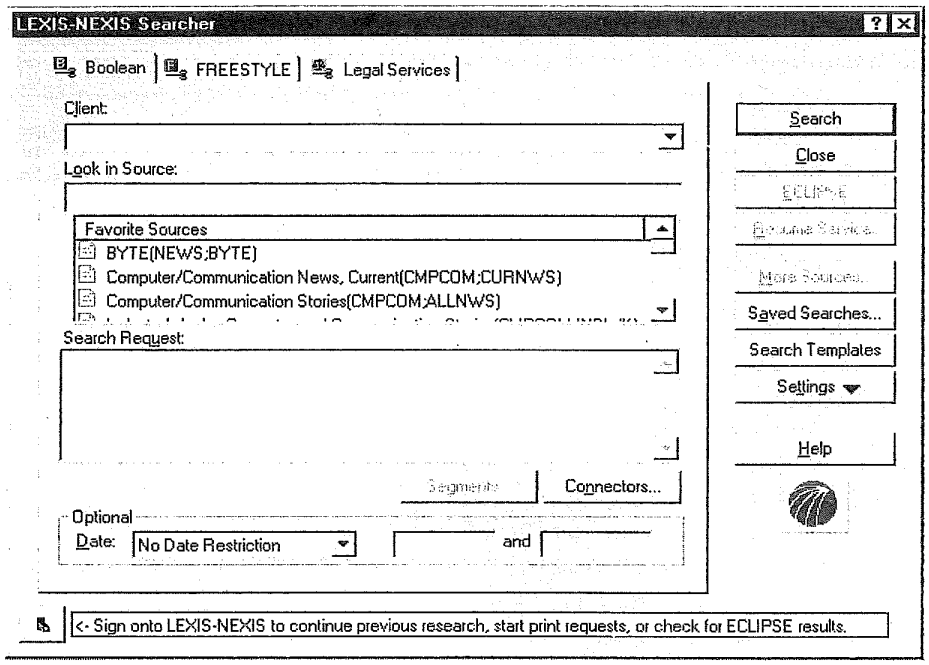


Figure 10.3 The LEXIS-NEXIS source selection screen.

learn through experience and years of training which sources are appropriate for various information needs. The restricted nature of traditional interfaces to information collections discourages exploration and discovery of new useful sources. However, recently researchers have devised a number of mechanisms to help users understand the contents of collections as a way of getting started in their search.

10.4.2 Overviews

Faced with a large set of text collections, how can a user choose which to begin with? One approach is to study an overview of the contents of the collections. An overview can show the topic domains represented within the collections, to help users select or eliminate sources from consideration. An overview can help users get started, directing them into general neighborhoods, after which they can navigate using more detailed descriptions. Shneiderman [724] advocates an interaction model in which the user begins with an overview of the information to be worked with, then pans and zooms to find areas of potential interest, and then view details. The process is repeated as often as necessary.

Three types of overviews are discussed in this subsection. The first is display and navigation of large topical *category hierarchies* associated with the documents of a collection. The second is automatically derived overviews, usually created by unsupervised *clustering techniques* on the text of documents, that attempt to extract overall characterizing themes from collections. The third type

of overview is that created by applying a variant of *co-citation analysis* on connections or links between different entities within a collection. Other kinds of overviews are possible, for example, showing graphical depictions of bookshelves or piles of books [681, 46].

Category or Directory Overviews

There exist today many large online text collections to which category labels have been assigned. Traditional online bibliographic systems have for decades assigned subject headings to books and other documents [752]. MEDLINE, a large collection of biomedical articles, has associated with it Medical Subject Headings (MeSH) consisting of approximately 18,000 categories [523]. The Association for Computing Machinery (ACM) has developed a hierarchy of approximately 1200 category (keyword) labels.† Yahoo![839], one of the most popular search sites on the World Wide Web, organizes Web pages into a hierarchy consisting of thousands of category labels.

The popularity of Yahoo! and other Web directories suggests that hierarchically structured categories are useful starting points for users seeking information on the Web. This popularity may reflect a preference to begin at a logical starting point, such as the home page for a set of information, or it may reflect a desire to avoid having to guess which words will retrieve the desired information. (It may also reflect the fact that directory services attempt to cull out low quality Web sites.)

The meanings of category labels differ somewhat among collections. Most are designed to help organize the documents and to aid in query specification. Unfortunately, users of online bibliographic catalogs rarely use the available subject headings [335, 222]. Hancock-Beaulieu and Drabenstott and Weller, among others, put much of the blame on poor (command line-based) user interfaces which provide little aid for selecting subject labels and require users to scroll through long alphabetic lists. Even with graphical Web interfaces, finding the appropriate place within a category hierarchy can be a time-consuming task, and once a collection has been found using such a representation, an alternative means is required for searching within the site itself.

Most interfaces that depict category hierarchies graphically do so by associating a document directly with the node of the category hierarchy to which it has been assigned. For example, clicking on a category link in Yahoo! brings up a list of documents that have been assigned that category label. Conceptually, the document is stored within the category label. When navigating the results of a search in Yahoo!, the user must look through a list of category labels and guess which one is most likely to contain references to the topic of interest. A wrong path requires backing up and trying again, and remembering which pages contain which information. If the desired information is deep in the hierarchy, or

† `http://www.acm.org/class`

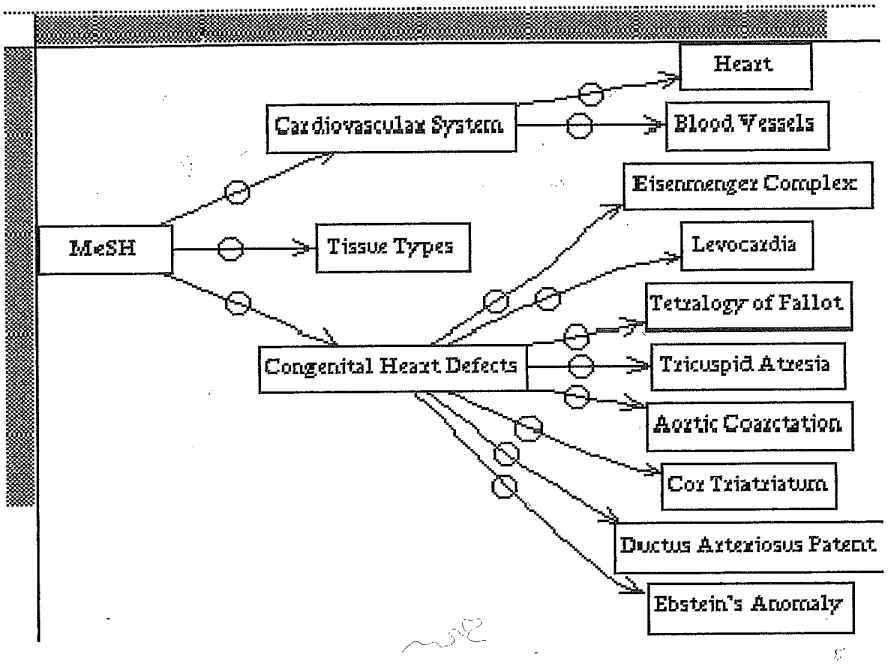


Figure 10.4 The MeSHBrowse interface for viewing category labels hierarchically [453].

not available at all, this can be a time-consuming and frustrating process. Because documents are conceptually stored 'inside' categories, users cannot create queries based on combinations of categories using this interface.

It is difficult to design a good interface to integrate category selection into query specification, in part because display of category hierarchies takes up large amounts of screen space. For example, Internet Grateful Med‡ is a Web-based service that allows an integration of search with display and selection of MeSH category labels. After the user types in the name of a potential category label, a long list of choices is shown in a page. To see more information about a given label, the user selects a link (e.g., Radiation Injuries). The causes the context of the query to disappear because a new Web page appears showing the ancestors of the term and its immediate descendants. If the user attempts to see the siblings of the parent term (Wounds and Injuries) then a new page appears that changes the context again. Radiation Injuries appears as one of many siblings and its children can no long be seen. To go back to the query, the illustration of the category hierarchy disappears.

The MeSHBrowse system [453] allows users to interactively browse a subset of semantically associated links in the MeSH hierarchy. From a given starting point, clicking on a category causes the associated categories to be displayed in a two-dimensional tree representation. Thus only the relevant subset of the

‡ http://igm.nlm.nih.gov:80/

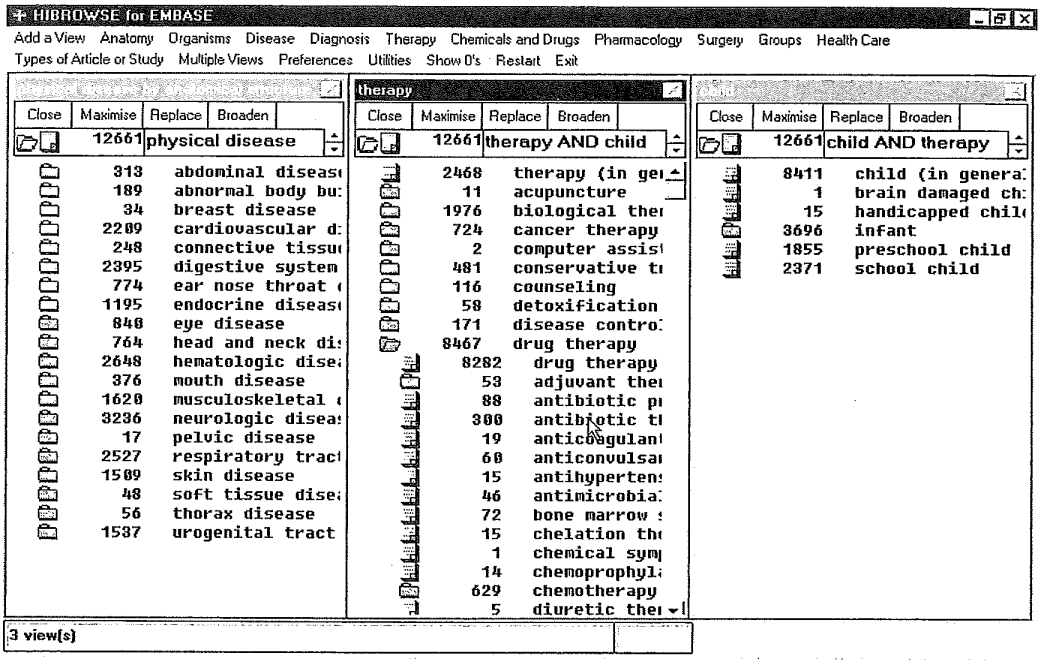


Figure 10.5 The HiBrowse interface for viewing category labels hierarchically and according to facets [646].

hierarchy is shown at one time, making browsing of this very large hierarchy a more tractable endeavor. The interface has the space limitations inherent in a two-dimensional hierarchy display and does not provide mechanisms for search over an underlying document collection. See Figure 10.4.

The HiBrowse system [646] represents category metadata more efficiently by allowing users to display several different subsets of category metadata simultaneously. The user first selects which attribute type (or facet, as attributes are called in this system) to display. For example, the user may first choose the 'physical disease' value for the Disease facet. The categories that appear one level below this are shown along with the number of documents that contain each category. The user can then select other attribute types, such as Therapy and Groups (by age). The number of documents that contain attributes from all three types are shown. If the user now selects a refinement of one of the categories, such as the 'child' value for the Groups attribute, then the number of documents that contain all three selected facet types are shown. At the same time, the number of documents containing the subcategories found below 'physical disease' and 'therapy (general)' are updated to reflect this more restricted specification. See Figure 10.5. A problem with the HiBrowse system is that it requires users to navigate through the category hierarchy, rather than specify queries directly. In other words, query specification is not tightly coupled with display of category metadata. As a solution to some of these problems, the Cat-a-Cone interface [358] will be described in section 10.8.

Automatically Derived Collection Overviews

Many attempts to display overview information have focused on automatically extracting the most common general themes that occur within the collection. These themes are derived via the use of unsupervised analysis methods, usually variants of document clustering. Clustering organizes documents into groups based on similarity to one another; the centroids of the clusters determine the themes in the collections.

The Scatter/Gather browsing paradigm [203, 202] clusters documents into topically-coherent groups, and presents descriptive textual summaries to the user. The summaries consist of topical terms that characterize each cluster generally, and a set of typical titles that hint at the contents of the cluster. Informed by the summaries, the user may select a subset of clusters that seem to be of most interest, and recluster their contents. Thus the user can examine the contents of each subcollection at progressively finer granularity of detail. The reclustering is computed on-the-fly; different themes are produced depending on the documents contained in the subcollection to which clustering is applied. The choice of clustering algorithm influences what clusters are produced, but no one algorithm has been shown to be particularly better than the rest when producing the same number of clusters [816].

A user study [640] showed that the use of Scatter/Gather on a large text collection successfully conveys some of the content and structure of the corpus. However, that study also showed that Scatter/Gather without a search facility was less effective than a standard similarity search for finding relevant documents for a query. That is, subjects allowed only to navigate, not to search over, a hierarchical structure of clusters covering the entire collection were less able to find documents relevant to the supplied query than subjects allowed to write queries and scan through retrieval results.

It is possible to integrate Scatter/Gather with conventional search technology by applying clustering on the results of a query to organize the retrieved documents (see Figure 10.6). An offline experiment [359] suggests that clustering may be more effective if used in this manner. The study found that documents relevant to the query tend to fall mainly into one or two out of five clusters, if the clusters are generated from the top-ranked documents retrieved in response to the query. The study also showed that precision and recall were higher within the best cluster than within the retrieval results as a whole. The implication is that a user might save time by looking at the contents of the cluster with the highest proportion of relevant documents and at the same time avoiding those clusters with mainly non-relevant documents. Thus, clustering of retrieval results may be useful for helping direct users to a subset of the retrieval results that contain a large proportion of the relevant documents.

General themes do seem to arise from document clustering, but the themes are highly dependent on the makeup of the documents within the clusters [359, 357]. The unsupervised nature of clustering can result in a display of topics at varying levels of description. For example, clustering a collection of documents about computer science might result in clusters containing documents about

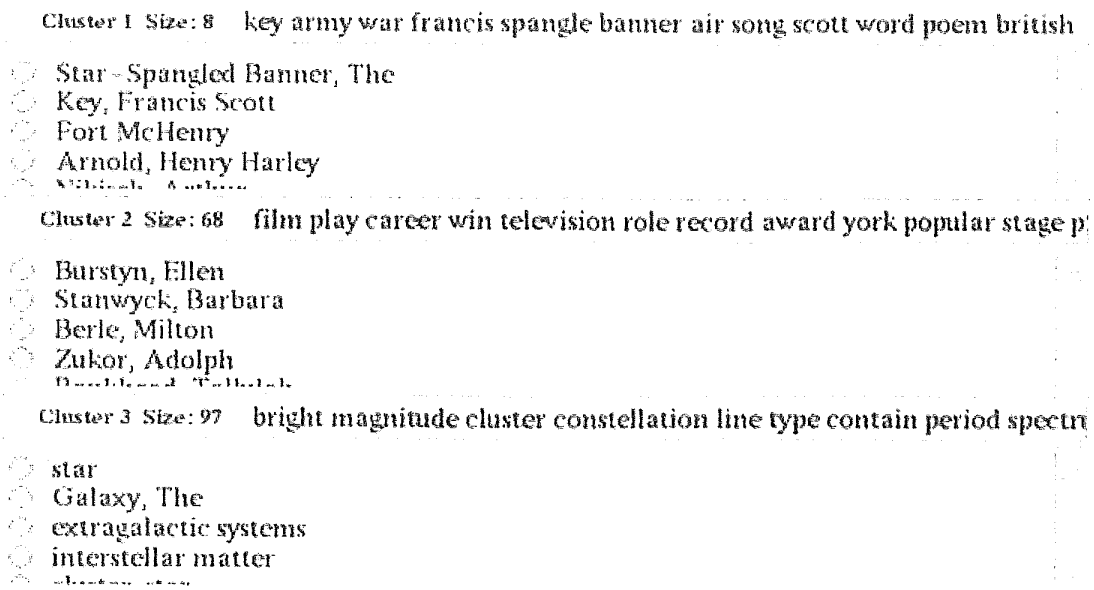


Figure 10.6 Display of Scatter/Gather clustering retrieval results [203].

artificial intelligence, computer theory, computer graphics, computer architecture, programming languages, government, and legal issues. The latter two themes are more general than the others, because they are about topics outside the general scope of computer science. Thus clustering can results in the juxtaposition of very different levels of description within a single display.

Scatter/Gather shows a textual representation of document clusters. Researchers have developed several approaches to map documents from their high dimensional representation in document space into a 2D representation in which each document is represented as a small glyph or icon on a map or within an abstract 2D space. The functions for transforming the data into the lower dimensional space differ, but the net effect is that each document is placed at one point in a scatter-plot-like representation of the space. Users are meant to detect themes or clusters in the arrangement of the glyphs. Systems employing such graphical displays include BEAD [156], the Galaxy of News [671], and ThemeScapes [821]. The ThemeScapes view imposes a three-dimensional representation on the results of clustering (see Figure 10.7). The layout makes use of 'negative space' to help emphasize the areas of concentration where the clusters occur. Other systems display inter-document similarity hierarchically [529, 14], while still others display retrieved documents in networks based on inter-document similarity [262, 761].

Kohonen's feature map algorithm has been used to create maps that graphically characterize the overall content of a document collection or subcollection [520, 163] (see Figure 10.8). The regions of the 2D map vary in size and shape corresponding to how frequently documents assigned to the corresponding themes occur within the collection. Regions are characterized by single words or phrases,

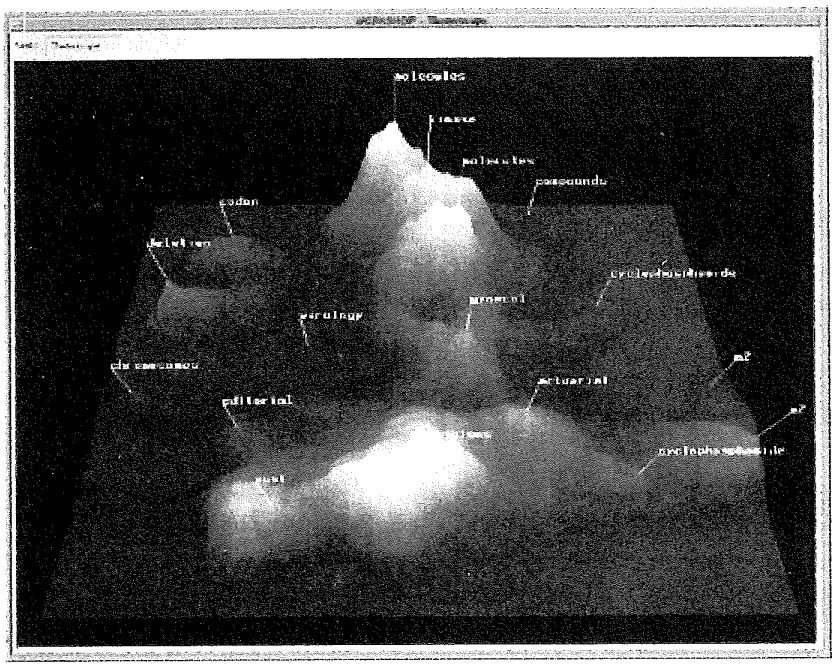


Figure 10.7 A three-dimensional overview based on document clustering [821].

and adjacency of regions is meant to reflect semantic relatedness of the themes within the collection. A cursor moved over a document region causes the titles of the documents most strongly associated with that region to be displayed in a pop-up window. Documents can be associated with more than one region.

Evaluations of Graphical Overviews

Although intuitively appealing, graphical overviews of large document spaces have yet to be shown to be useful and understandable for users. In fact, evaluations that have been conducted so far provide negative evidence as to their usefulness. One study found that for non-expert users the results of clustering were difficult to use, and that graphical depictions (for example, representing clusters with circles and lines connecting documents) were much harder to use than textual representations (for example, showing titles and topical words, as in Scatter/Gather), because documents' contents are difficult to discern without actually reading some text [443].

Another recent study compared the Kohonen feature map overview representation on a browsing task to that of Yahoo! [163]. For one of the tasks, subjects were asked to find an 'interesting' Web page within the entertainment category of Yahoo! and of an organization of the same Web pages into a Kohonen map layout. The experiment varied whether subjects started in Yahoo! or in the graphical map. After completion of the browing task, subjects were asked to attempt to repeat the browse using the other tool. For the subjects that

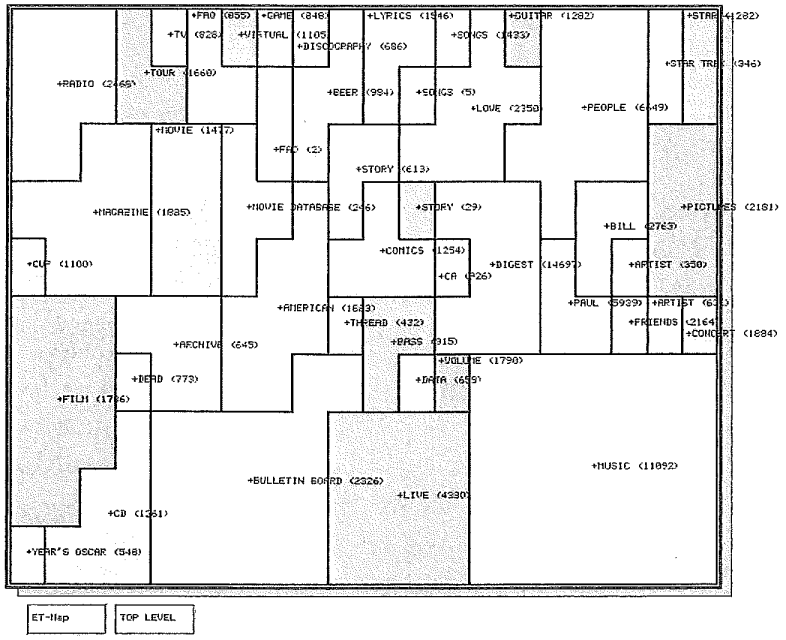


Figure 10.8 A two-dimensional overview created using a Kohonen feature map learning algorithm on Web pages having to do with the topic Entertainment [163].

began with the Kohonen map visualization, 11 out of 15 found an interesting page within ten minutes. Eight of these were able to find the same page using Yahoo!. Of the subjects who started with Yahoo!, 14 out of 16 were able to find interesting home pages. However, only two of the 14 were able to find the page in the graphical map display! This is strong evidence against the navigability of the display and certainly suggests that the simple label view provided by Yahoo! is more useful. However, the map display may be more useful if the system is modified to tightly integrate querying with browsing.

The subjects did prefer some aspects of the map representation. In particular, some liked the ease of being able to jump from one area to another without having to back up as is required in Yahoo!, and some liked the fact that the maps have varying levels of granularity. The subjects disliked several aspects of the display. The experimenters found that some subjects expressed a desire for a visible hierarchical organization, others wanted an ability to zoom in on a subarea to get more detail, and some users disliked having to look through the entire map to find a theme, desiring an alphabetical ordering instead. Many found the single-term labels to be misleading, in part because they were ambiguous (one region called 'BILL' was thought to correspond to a person's name rather than counting money).

The authors concluded that this interface is more appropriate for casual browsing than for search. In general, unsupervised thematic overviews are perhaps most useful for giving users a 'gist' of the kinds of information that can be

found within the document collection, but generally have not been shown to be helpful for use in the information access process.

Co-citation Clustering for Overviews

Citation analysis has long been recognized as a way to show an overview of the contents of a collection [812]. The main idea is to determine 'centrally-located' documents based on co-citation patterns. There are different ways to determine citation patterns: one method is to measure how often two articles are cited together by a third. Another alternative is to pair articles that cite the same third article. In both cases the assumption is that the paired articles share some commonalities. After a matrix of co-citations is built, documents are clustered based on the similarity of their co-citation patterns. The resulting clusters are interpreted to indicate dominant themes within the collection. Clustering can focus on the authors of the documents rather than the contents, to attempt to identify central authors within a field. This idea has recently been implemented using Web-based documents in the Referral Web project [432]. The idea has also been applied to Web pages, using Web link structure to identify major topical themes among Web pages [485, 639]. A similar idea, but computed a different way, is used to explicitly identify pages that act as good starting points for particular topics (called 'authority pages' by Kleinberg [444]).

10.4.3 Examples, Dialogs, and Wizards

Another way to help users get started is to start them off with an example of interaction with the system. This technique is also known as *retrieval by reformulation.* An early version of this idea is embodied in the Rabbit system [818] which provides graphical representations of example database queries. A general framework for a query is shown to the user who then modifies it to construct a partially complete description of what they want. The system then shows an example of the kind of information available that matches this partial description. For instance, if a user searching a computer products database indicates an interest in disks, an example item is retrieved with its disk descriptors filled in. The user can use or modify the displayed descriptors, and iterate the procedure.

The idea of retrieval by reformulation has been developed further and extended to the domains of user interface development [581] and software engineering [669]. The Helgon system [255] is a modern variant of this idea applied to bibliographic database information. In Helgon, users begin by navigating a hierarchy of topics from which they select structured examples, according to their interests. If a feature of an example is inappropriately set, the user can modify the feature to indicate how it would appear in the desired information. Unfortunately, in tests with users, the system was found to be problematic. Users had problems with the organization of the hierarchy, and found that searching for a useful example by critiquing an existing one to be tedious. This result

underscores an unfortunate difficulty with examples and dialogues: that of getting the user to the right starting dialogue or the right example strategy becomes a search problem in itself. (How to index prior examples is studied extensively in the case-based reasoning (CBR) literature [492, 449].)

A more dynamic variation on this theme is the interactive dialog. Dialog-based interfaces have been explored since the early days of information retrieval research, in an attempt to mimic the interaction provided by a human search intermediary (e.g., a reference librarian). Oddy did early work in the THOMAS system, which provided a question and answer session within a command-line-based interface [615]. More recently, Belkin *et al.* have defined quite elaborate dialog interaction models [75] although these have not been assessed empirically to date.

The DLITE system interface [192] uses an animated focus-plus-context dialog as a way to acquaint users with standard sequences of operations within the system. Initially an outline of all of the steps of the dialog is shown as a list. The user can expand the explanation of any individual step by clicking on its description. The user can expand out the entire dialog to see what questions are coming next, and then collapse it again in order to focus on the current tactic.

A more restricted form of dialog that has become widely used in commercial products is that of the *wizard.* This tool helps users in time-limited tasks, but does not attempt to overtly teach the processes required to complete the tasks. The wizard presents a step-by-step shortcut through the sequence of menu choices (or tactics) that a user would normally perform in order to get a job done, reducing user input to just a few choices with default settings [636]. A recent study [145] found wizards to be useful for goals that require many steps, for users who lack necessary domain knowledge (for example, a restaurant owner installing accounting software), and when steps must be completed in a fixed sequence (for example, a procedure for hiring personnel). Properties of successful wizards included allowing users to rerun a wizard and modify their previous work, showing an overview of the supported functions, and providing lucid descriptions and understandable outcomes for choices. Wizards were found not to be helpful when the interface did not solve a problem effectively (for example, a commercial wizard for setting up a desktop search index requests users to specify how large to make the index, but supplies no information about how to make this decision). Wizards were also found not to be helpful when the goal was to teach the user how to use the interface, and when the wizard was not user-tested. It maybe the case that information access is too variable a process for the use of wizards.

A *guided tour* leads a user through a sequence of navigational choices through hypertext links, presenting the nodes in a logical order for some goal. In a dynamic tour, only relevant nodes are displayed, as opposed to the static case where the author decides what is relevant before the users have even formulated their queries [329]. A recent application is the Walden Paths project which enables teachers to define instructionally useful paths through pages found on the Web [289]. This approach has not been commonly used to date for

information access but could be a promising direction for acquainting users with search strategies in large hyperlinked systems.

10.4.4 Automated Source Selection

Human-computer interfaces for helping guide users to appropriate sources is a wide open area for research. It requires both eliciting the information need from users and understanding which needs can be satisfied by which sources. An ambitious approach is to build a model of the source and of the information need of the user and try to determine which fit together best. User modeling systems and intelligent tutoring systems attempt to do this both for general domains [204, 814] and for online help systems [378].

A simpler alternative is to create a representation of the contents of information sources and match this representation against the query specification. This approach is taken by GlOSS, a system which tries to determine in advance the best bibliographic database to send a search request to, based on the terms in the query [765]. The system uses a simple analysis of the combined frequencies of the query words within the individual collections. The SavvySearch system [383] takes this idea a step further, using actions taken by users after a query to decide how to decrease or increase the ranking of a search engine for a particular query (see also Chapter 13).

The flip side to automatically selecting the best source for a query is to automatically send a query to multiple sources and then combine the results from the various systems in some way. Many metasearch engines exist on the Web. How to combine the results effectively is an active area of research, sometimes known as collection fusion [63, 767, 388].

10.5 Query Specification

To formulate a query, a user must select collections, metadata descriptions, or information sets against which the query is to be matched, and must specify words, phrases, descriptors, or other kinds of information that can be compared to or matched against the information in the collections. As a result, the system creates a set of documents, metadata, or other information type that match the query specification in some sense and displays the results to the user in some form.

Shneiderman [725] identifies five primary human-computer interaction styles. These are: *command language*, *form fillin*, *menu selection*, *direct manipulation*, and *natural language*.§ Each technique has been used in query specification interfaces and each has advantages and disadvantages. These are described below in the context of Boolean query specification.

§ This list omits non-visual modalities, such as audio.

10.5.1 Boolean Queries

In modern information access systems the matching process usually employs a statistical ranking algorithm. However, until recently most commercial full-text systems and most bibliographic systems supported only Boolean queries. Thus the focus of many information access studies has been on the problems users have in specifying Boolean queries. Unfortunately, studies have shown time and again that most users have great difficulty specifying queries in Boolean format and often misjudge what the results will be [111, 322, 558, 841].

Boolean queries are problematic for several reasons. Foremost among these is that most people find the basic syntax counter-intuitive. Many English-speaking users assume everyday semantics are associated with Boolean operators when expressed using the English words AND and OR, rather than their logical equivalents. To inexperienced users, using AND implies the widening of the scope of the query, because more kinds of information are being requested. For instance, 'dogs and cats' may imply a request for documents about dogs and documents about cats, rather than documents about both topics at once. 'Tea or coffee' can imply a mutually exclusive choice in everyday language. This kind of conceptual problem is well documented [111, 322, 558, 841]. In addition, most query languages that incorporate Boolean operators also require the user to specify complex syntax for other kinds of connectors and for descriptive metadata. Most users are not familiar with the use of parentheses for nested evaluation, nor with the notions associated with operator precedence.

By serving a massive audience possessing little query-specification experience, the designers of World Wide Web search engines have had to come up with more intuitive approaches to query specification. Rather than forcing users to specify complex combinations of ANDs and ORs, they allow users to choose from a selection of common simple ways of combining query terms, including 'all the words' (place all terms in a conjunction) and 'any of the words' (place all terms in a disjunction).

Another Web-based solution is to allow syntactically-based query specification, but to provide a simpler or more intuitive syntax. The '+' prefix operator gained widespread use with the advent of its use as a mandatory specifier in the Altavista Web search engine. Unfortunately, users can be misled to think it is an infix AND rather than a prefix mandatory operator, and thus assume that 'cat + dog' will only retrieve articles containing both terms (where in fact this query requires dog but allows cat to be optional).

Another problem with pure Boolean systems is they do not rank the retrieved documents according to their degree of match to the query. In the pure Boolean framework a document either satisfies the query or it does not. Commercial systems usually resort to ordering documents according to some kind of descriptive metadata, usually in reverse chronological order. (Since these systems usually index timely data corresponding to newspaper and news wires, date of publication is often one of the most salient features of the document.) Web-based systems usually rank order the results of Boolean queries using statistical algorithms and Web-specific heuristics.

10.5.2 From Command Lines to Forms and Menus

Aside from conceptual misunderstandings of the logical meaning of AND and OR, another part of the problem with pure Boolean query specification in online bibliographic systems is the arbitrariness of the syntax and the contextlessness nature of the TTY-based interface in which they are predominantly available. Typically input is typed at a prompt and is of a form something like the following:

> COMMAND ATTRIBUTE value {BOOLEAN-OPERATOR ATTRIBUTE value}*
> e.g.,
> FIND PA darwin AND TW species OR TW descent
> or
> FIND TW Mt St. Helens AND DATE 1981

(These examples are derived from the syntax of the telnet interface to the University of California Melvyl system [526].) The user must remember the commands and attribute names, which are easily forgotten between usages of the system [553]. Compounding this problem, despite the fact that the command languages for the two main online bibliographic systems at UC Berkeley have different but very similar syntaxes, after more than ten years one of the systems still reports an error if the author field is specified as PA instead of PN, as is done in the other system. This lack of flexibility in the syntax is characteristic of interfaces designed to suit the system rather than its users.

The new Web-based version of Melvyl‖ provides form fillin and menu selection so the user no longer has to remember the names and types of attributes available. Users select metadata types from listboxes and attributes are shown explicitly, allowing selection as an alternative to specification. For example, the 'search type' field is adjacent to an entry form in which users can enter keywords, and a choice between AND and NOT is provided adjacent to a list of the available document types (editorial, feature, etc.). Only the metadata associated with a given collection is shown in the context of search over that collection. (Unfortunately the system is restricted to searching over only one database at a time. It does however provide a mechanism for applying a previously executed search to a new database.) See Figure 10.9.

The Web-based version of Melvyl also allows retention of context between searches, storing prior results in tables and hyperlinking these results to lists containing the retrieved bibliographic information. Users can also modify any of the previously submitted queries by selecting a checkbox beside the record of the query. The graphical display makes explicit and immediate many of the powerful options of the system that most users would not learn using the command-line version of the interface.

Bit-mapped displays are an improvement over command-line interface, but do not solve all the problems. For example, a blank entry form is in some ways

‖ http://www.melvyl.ucop.edu/

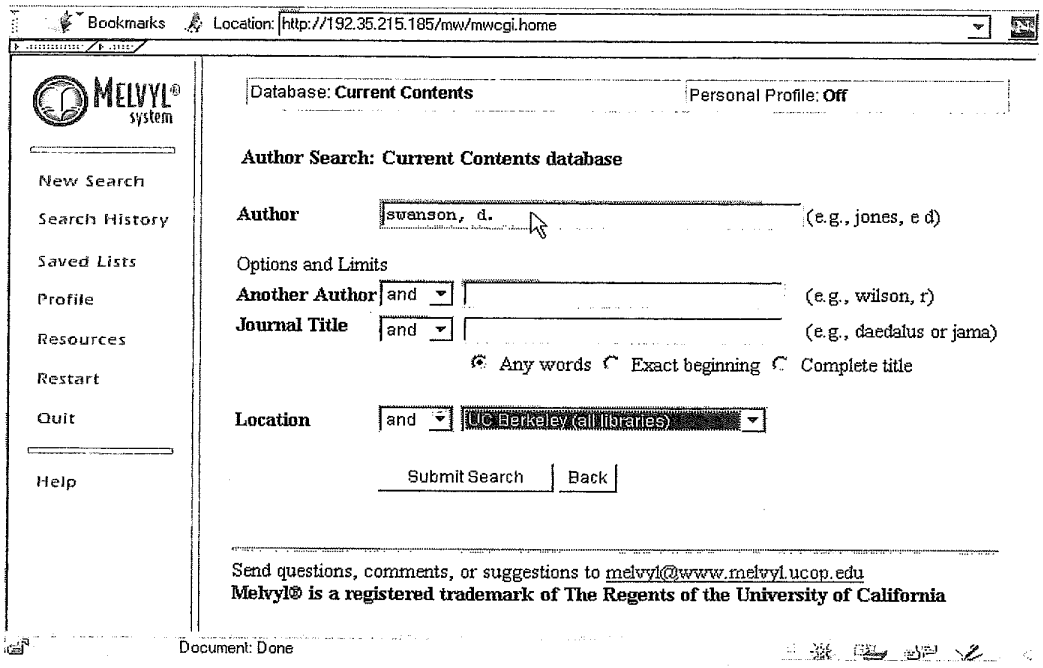


Figure 10.9 A view of query specification in the Web-based version of the Melvyl bibliographic catalog. Copyright © 1998, The Regents of the University of California.

not much better than a TTY prompt, because it does not provide the user with clues about what kinds of terms should be entered.

10.5.3 Faceted Queries

Yet another problem with Boolean queries is that their strict interpretation tends to yield result sets that are either too large, because the user includes many terms in a disjunct, or are empty, because the user conjoins terms in an effort to reduce the result set. This problem occurs in large part because the user does not know the contents of the collection or the role of terms within the collection.

A common strategy for dealing with this problem, employed in systems with command-line-based interfaces like DIALOG's, is to create a series of short queries, view the number of documents returned for each, and combine those queries that produce a reasonable number of results. For example, in DIALOG, each query produces a resulting set of documents that is assigned an identifying name. Rather than returning a list of titles themselves, DIALOG shows the set number with a listing of the number of matched documents. Titles can be shown by specifying the set number and issuing a command to show the titles. Document sets that are not empty can be referred to by a set name and combined with AND operations to produce new sets. If this set in turn is too small, the user can back up and try a different combination of sets, and this process is repeated in pursuit of producing a reasonably sized document set.

This kind of query formulation is often called a *faceted* query, to indicate that the user's query is divided into topics or facets, each of which should be

present in the retrieved documents [553, 348]. For example, a query on drugs for the prevention of osteoporosis might consist of three facets, indicated by the disjuncts

(osteoporosis OR 'bone loss')
(drugs OR pharmaceuticals)
(prevention OR cure)

This query implies that the user would like to view documents that contain all three topics.

A technique to impose an ordering on the results of Boolean queries is what is known as *post-coordinate* or *quorum-level* ranking [700, Ch. 8]. In this approach, documents are ranked according to the size of the subset of the query terms they contain. So given a query consisting of 'cats,' 'dogs,' 'fish,' and 'mice,' the system would rank a document with at least one instance of 'cats,' 'dogs,' and 'fish' higher than a document containing 30 occurrences of 'cats' but no occurrences of the other terms.

Combining faceted queries with quorum ranking yields a situation intermediate between full Boolean syntax and free-form natural language queries. An interface for specifying this kind of interaction can consist of a list of entry lines. The user enters one topic per entry line, where each topic consists of a list of semantically related terms that are combined in a disjunct. Documents that contain at least one term from each facet are ranked higher than documents containing terms only from one or a few facets. This helps ensure that documents which contain discussions of several of the user's topics are ranked higher than those that contain only one topic. By only requiring that one term from each facet be matched, the user can specify the same concept in several different ways in the hopes of increasing the likelihood of a match. If combined with graphical feedback about which subsets of terms matched the document, the user can see the results of a quorum ranking by topic rather than by word. Section 10.6 describes the TileBars interface which provides this type of feedback.

This idea can be extended yet another step by allowing users to weight each facet. More likely to be readily usable, however, is a default weighting in which the facet listed highest is assigned the most weight, the second facet is assigned less weight, and so on, according to some distribution over weights.

10.5.4 Graphical Approaches to Query Specification

Direct manipulation interfaces provide an alternative to command-line syntax. The properties of direct manipulation are [725, p.205]: (1) continuous representation of the object of interest, (2) physical actions or button presses instead of complex syntax, and (3) rapid incremental reversible operations whose impact on the object of interest is immediately visible. Direct manipulation interfaces often evoke enthusiasm from users, and for this reason alone it is worth exploring their use. Although they are not without drawbacks, they are easier to use than other methods for many users in many contexts.

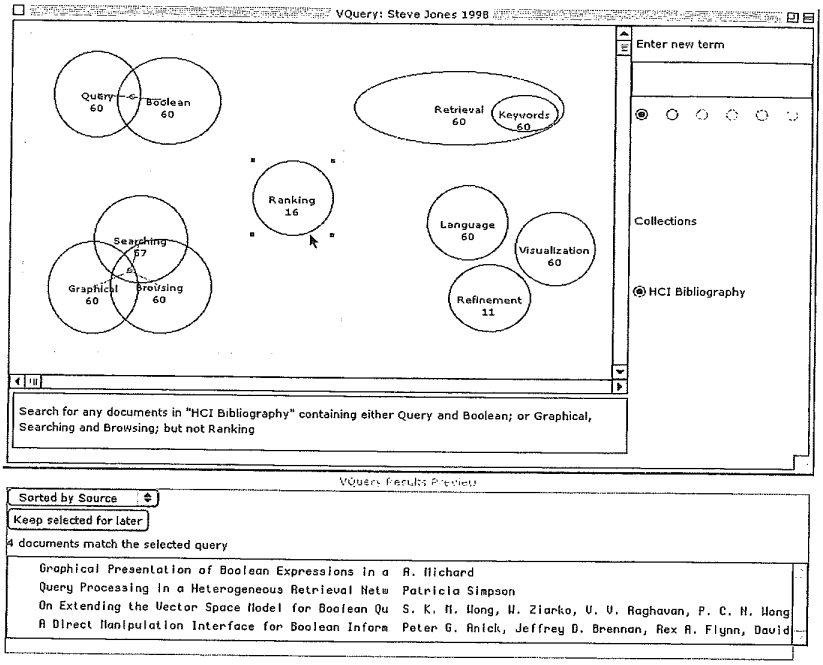


Figure 10.10 The VQuery Venn diagram visualization for Boolean query specification [417].

Several variations of graphical interfaces, both directly manipulable and static, have been developed for simplifying the specification of Boolean syntax. User studies tend to reveal that these graphical interfaces are more effective in terms of accuracy and speed than command-language counterparts. Three such approaches are described below.

Graphical depictions of *Venn diagrams* have been proposed several times as a way to improve Boolean query specification. A query term is associated with a ring or circle and intersection of rings indicates conjunction of terms. Typically the number of documents that satisfy the various conjuncts are displayed within the appropriate segments of the diagram. Several studies have found such interfaces more effective than their command-language-based syntax [417, 368, 558]. Hertzum and Frokjaer found that a simple Venn diagram representation produced faster and more accurate results than a Boolean query syntax. However, a problem with this format is the limitations on the complexity of the expression. For example, a maximum of three query terms can be ANDed together in a standard Venn diagram. Innovations have been designed to get around this problem, as seen in the VQuery system [417] (see Figure 10.10). In VQuery, a direct manipulation interface allows users to assign any number of query terms to ovals. If two or more ovals are placed such that they overlap with one another, and if the user selects the area of their intersection, an AND is implied among those terms. (In Figure 10.10, the term 'Query' is conjoined with 'Boolean'.) If the user selects outside the area of intersection but within the ovals, an OR is implied among the corresponding terms. A NOT operation

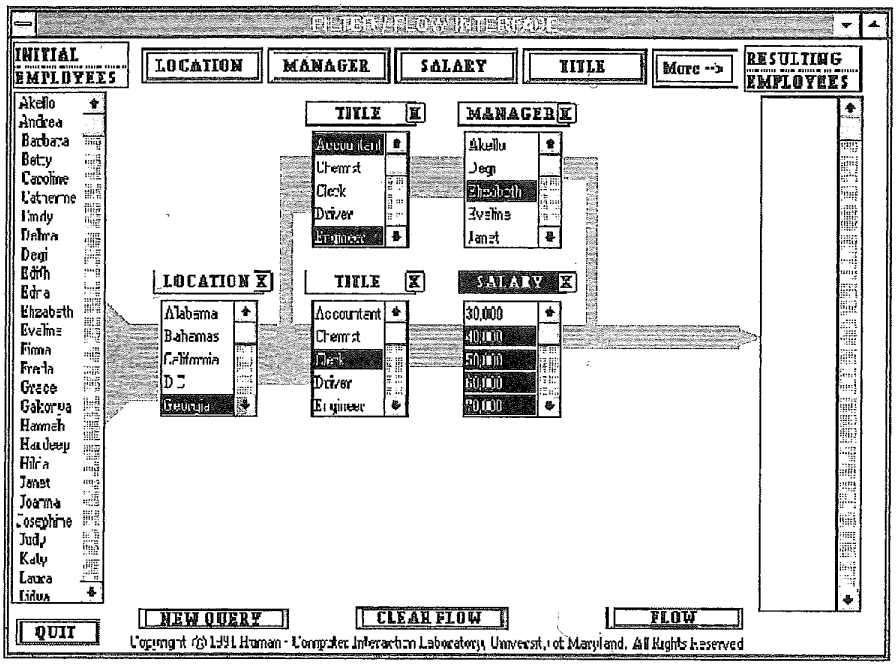


Figure 10.11 The filter-flow visualization for Boolean query specification [841].

is associated with any term whose oval appears in the active area of the display but which remains unselected (in the figure, NOT 'Ranking' has been specified). An active area indicates the current query; all groups of ovals within the active area are considered part of a conjunction. Ovals containing query terms can be moved out of the active area for later use.

Young and Shneiderman [841] found improvements over standard Boolean syntax by providing users with a direct manipulation *filter-flow* model. The user is shown a scrollable list of attribute types on the left-hand side and selects attributes from another list of attribute types shown across the top of the screen. Clicking on an attribute name causes a listbox containing values for those attributes to be displayed in the main portion of the screen. The user then selects which values of the attributes to let the flow go through. Placing two or more of these attributes in sequence creates the semantics of a conjunct over the selected values. Placing two or more of these in parallel creates the semantics of a disjunct. The number of documents that match the query at each point is indicated by the width of the 'water' flowing from one attribute to the next. (See Figure 10.11.) A conjunct can reduce the amount of flow. The items that match the full query are shown on the far right-hand side. A user study found that fewer errors were made using the filter flow model than a standard SQL database query. However, the examples and study pertain only to database querying rather than information access, since the possible query terms for information access cannot be represented realistically in a scrollable list. This interface could perhaps be modified to better suit information access applications by having the user supply initial query terms, and using the attribute selection facility to show those terms

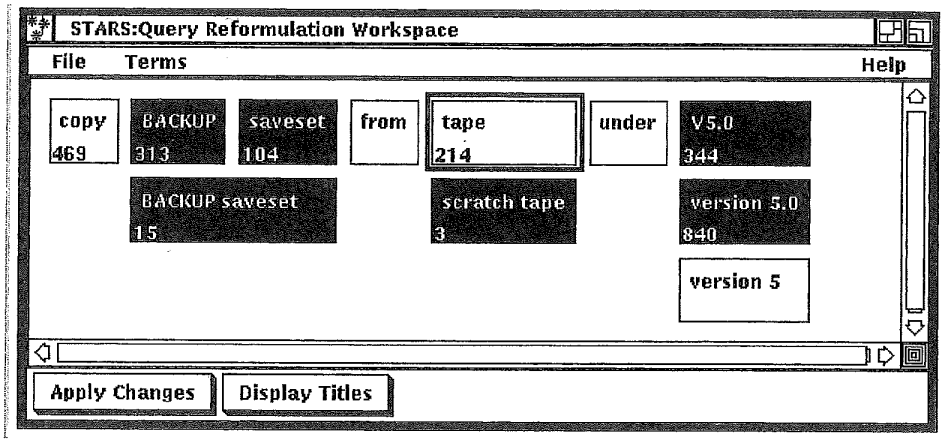


Figure 10.12 A block-oriented diagram visualization for Boolean query specification [21].

that are conceptually related to the query terms. Another alternative is to use this display as a category metadata selection interface (see Section 10.4).

Anick *et al.* [21] describe another innovative direct manipulation interface for Boolean queries. Initially the user types a natural language query which is automatically converted to a representation in which each query term is represented within a block. The blocks are arranged into rows and columns (See Figure 10.12). If two or more blocks appear along the same row they are considered to be ANDed together. Two or more blocks within the same column are ORed. Thus the user can represent a technical term in multiple ways within the same query, providing a kind of faceted query interface. For example, the terms 'version 5', 'version 5.0', and 'v5' might be shown in the same column. Users can quickly experiment with different combinations of terms within Boolean queries simply by activating and deactivating blocks. This facility also allows users to have multiple representations of the same term in different places throughout the display, thus allowing rapid feedback on the consequences of specifying various combinations of query terms. Informal evaluation of the system found that users were able to learn to manipulate the interface quickly and enjoyed using it. It was not formally compared to other interaction techniques [21].

This interface provides a kind of *query preview*: a low cost, rapid turnaround visualization of the results of many variations on a query [643]. Another example of query previewing can be found in some help systems, which show all the words in the index whose first letters match the characters that the user has typed so far. The more characters typed, the fewer possible matches become available. The HiBrowse system described above [646] also provides a kind of preview for viewing category hierarchies and facets, showing how many documents would be matched if a category one level below the current one were selected. It perhaps could be improved by showing the consequences of more combinations of categories in an animated manner. If based on prior action and interests of the user, query previewing may become more generally applicable for information access interfaces.

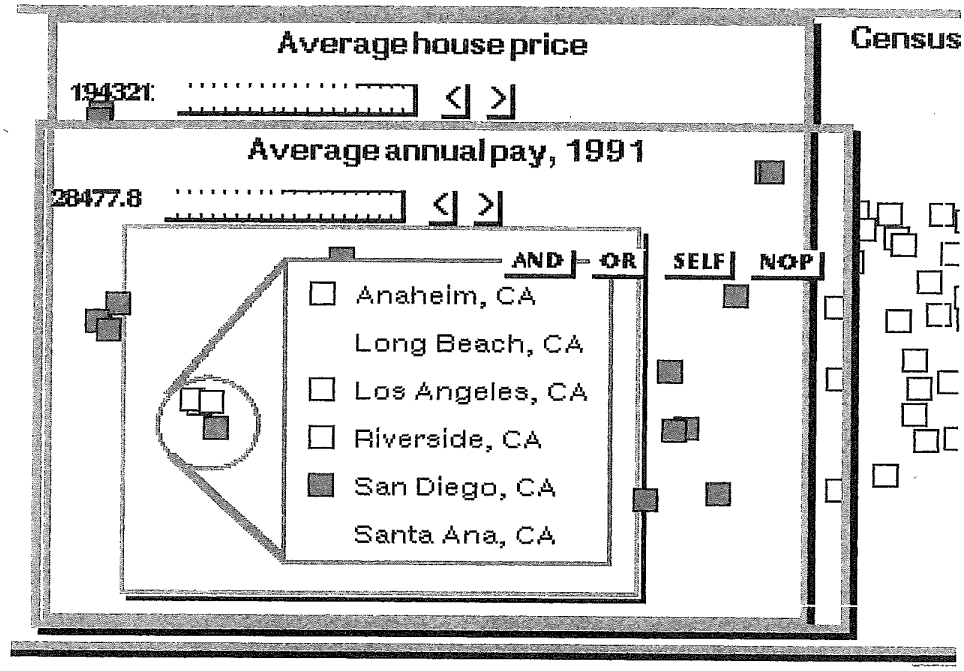


Figure 10.13 A magic lens interface for query specification (courtesy of Ken Fishkin).

A final example of a graphical approach to query specification is the use of magic lenses. Fishkin and Stone have suggested an extension to the usage of this visualization tool for the specification of Boolean queries [256]. Information is represented as lists or icons within a 2D space. Lenses act as filters on the document set. (See Figure 10.13.) For example, a word can be associated with a transparent lens. When this lens is placed over an iconic representation of a set of documents, it can cause all documents that do not contain a given word to disappear. If a second lens representing another word is then laid over the first, the lenses combine to act as a conjunction of the two words with the document set, hiding any documents that do not contain both words. Additional information can be adjusted dynamically, such as a minimum threshold for how often the term occurs in the documents, or an on-off switch for word stemming. For example, Figure 10.13 shows a disjunctive query that finds cities with relatively low housing prices or high annual salaries. One lens 'calls out' a clump of southern California cities, labeling each. Above that is a lens screening for cities with average house price below $194,321 (the data is from 1990), and above this one is a lens screening for cities with average annual pay above $28,477. This approach, while promising, has not been evaluated in an information access setting.

10.5.5 Phrases and Proximity

In general, proximity information can be quite effective at improving precision of searches. On the Web, the difference between a single-word query and a two-word

exact phrase match can mean the difference between an unmanageable mess of retrieved documents and a short list with mainly relevant documents.

A large number of methods for specifying phrases have been developed. The syntax in LEXIS-NEXIS requires the proximity range to be specified with an infix operator. For example, 'white w/3 house' means 'white within 3 words of house, independent of order.' Exact proximity of phrases is specified by simply listing one word beside the other, separated by a space. A popular method used by Web search engines is the enclosure of the terms between quotation marks. Shneiderman *et al.* [726] suggest providing a list of entry labels, as suggested above for specifying facets. The difference is, instead of a disjunction, the terms on each line are treated as a phrase. This is suggested as a way to guide users to more precise query specification.

The disadvantage of these methods is that they require exact match of phrases, when it is often the case (in English) that one or a few words comes between the terms of interest. For example, in most cases the user probably wants 'president' and 'lincoln' to be adjacent, but still wants to catch cases of the sort 'President Abraham Lincoln.' Another consideration is whether or not stemming is performed on the terms included in the phrase. The best solution may be to allow users to specify exact phrases but treat them as small proximity ranges, with perhaps an exponential fall-off in weight in terms of distance of the terms. This has been shown to be a successful strategy in non-interactive ranking algorithms [174]. It has also been shown that a combination of quorum ranking of faceted queries with the restriction that the facets occur within a small proximity range can dramatically improve precision of results [356, 566].

10.5.6 Natural Language and Free Text Queries

Statistical ranking algorithms have the advantage of allowing users to specify queries naturally, without having to think about Boolean or other operators. But they have the drawback of giving the user less feedback about and control over the results. Usually the result of a statistical ranking is the listing of documents and the association of a score, probability, or percentage beside the title. Users are given little feedback about why the document received the ranking it did and what the roles of the query terms are. This can be especially problematic if the user is particularly interested in one of the query terms being present.

One search strategy that can help with this particular problem with statistical ranking algorithms is the specification of 'mandatory' terms within the natural language query. This in effect helps the user control which terms are considered important, rather than relying on the ranking algorithm to correctly weight the query terms. But knowing to include a mandatory specification requires the user to know about a particular command and how it works.

The preceding discussion assumes that a natural language query entered by the user is treated as a bag of words, with stopwords removed, for the purposes of document match. However, some systems attempt to parse natural language queries in order to extract concepts to match against concepts in the

text collection [399, 552, 748].

Alternatively, the natural language syntax of a question can be used to attempt to answer the question. (Question answering in information access is different than that of database management systems, since the information desired is encoded within the text of documents rather than specified by the database schema.) The Murax system [463] determines from the syntax of a question if the user is asking for a person, place, or date. It then attempts to find sentences within encyclopedia articles that contain noun phrases that appear in the question, since these sentences are likely to contain the answer to the question. For example, given the question 'Who was the Pulitzer Prize-winning novelist that ran for mayor of New York City?,' the system extracts the noun phrases 'Pulitzer Prize,' 'winning novelist,' 'mayor,' and 'New York City.' It then looks for proper nouns representing people's names (since this is a 'who' question) and finds, among others, the following sentences:

> The Armies of the Night (1968), a personal narrative of the 1967 peace march on the Pentagon, won **Mailer** the **Pulitzer Prize** and the National Book Award.
>
> In 1969 **Mailer** ran unsuccessfully as an independent candidate for **mayor** of **New York City**.

Thus the two sentences link together the relevant noun phrases and the system hypothesizes (correctly) from the title of the article in which the sentences appear that Norman Mailer is the answer.

Another approach to automated question answering is the FAQ finder system which matches question-style queries against question-answer pairs on various topics [130]. The system uses a standard IR search to find the most likely FAQ (frequently asked questions) files for the question and then matches the terms in the question against the question portion of the question-answer pairs.

A less automated approach to question answering can be found in the Ask Jeeves system [34]. This system makes use of hand-picked Web sites and matches these to a predefined set of question types. A user's query is first matched against the question types. The user selects the most accurate rephrase of their question and this in turn is linked to suggested Web sites. For example, the question 'Who is the leader of Sudan?' is mapped into the question type 'Who is the head of state of X (Sudan)?' where the variable is replaced by a listbox of choices, with Sudan the selected choice in this case. This is linked to a Web page that lists current heads of state. The system also automatically substitutes in the name 'Sudan' in a query against that Web page, thus bringing the answer directly to the user's attention. The question is also sent to standard Web search engines. However, a system is only as good as its question templates. For example a question 'Where can I find reviews of spas in Calistoga?' matches the question 'Where can I find X (reviews) of activities for children aged Y (1)?' and 'Where can I find a concise encyclopedia article on X (hot springs)?'

10.6 Context

This section discusses interface techniques for placing the current document set in the *context* of other information types, in order to make the document set more understandable. This includes showing the relationship of the document set to query terms, collection overviews, descriptive metadata, hyperlink structure, document structure, and to other documents within the set.

10.6.1 Document Surrogates

The most common way to show results for a query is to list information about documents in order of their computed relevance to the query. Alternatively, for pure Boolean ranking, documents are listed according to a metadata attribute, such as date. Typically the document list consists of the document's title and a subset of important metadata, such as date, source, and length of the article. In systems with statistical ranking, a numerical score or percentage is also often shown alongside the title, where the score indicates a computed degree of match or probability of relevance. This kind of information is sometimes referred to as a *document surrogate.* See Figure 10.14 from [824].

Some systems provide users with a choice between a short and a detailed view. The detailed view typically contains a summary or abstract. In bibliographic systems, the author-written or service-written abstract is shown. Web search engines automatically generate excerpts, usually extracting the first few lines of non-markup text in the Web page.

In most interfaces, clicking on the document's title or an iconic representation of the document shown beside the title will bring up a view of the document itself, either in another window on the screen, or replacing the listing of search results. (In traditional bibliographic systems, the full text was unavailable online, and only bibliographic records could be readily viewed.)

10.6.2 Query Term Hits Within Document Content

In systems in which the user can view the full text of a retrieved document, it is often useful to *highlight* the occurrences of the terms or descriptors that match those of the user's query. It can also be useful for the system to scroll the view of the document to the first passage that contains one or more of the query terms, and highlight the matched terms in a contrasting color or reverse video. This display is thought to help draw the user's attention to the parts of the document most likely to be relevant to the query. Highlighting of query terms has been found time and again to be a useful feature for information access interfaces [481],[542, p.31]. Color highlighting has also recently been found to be useful for scanning lists of bibliographic records [52].

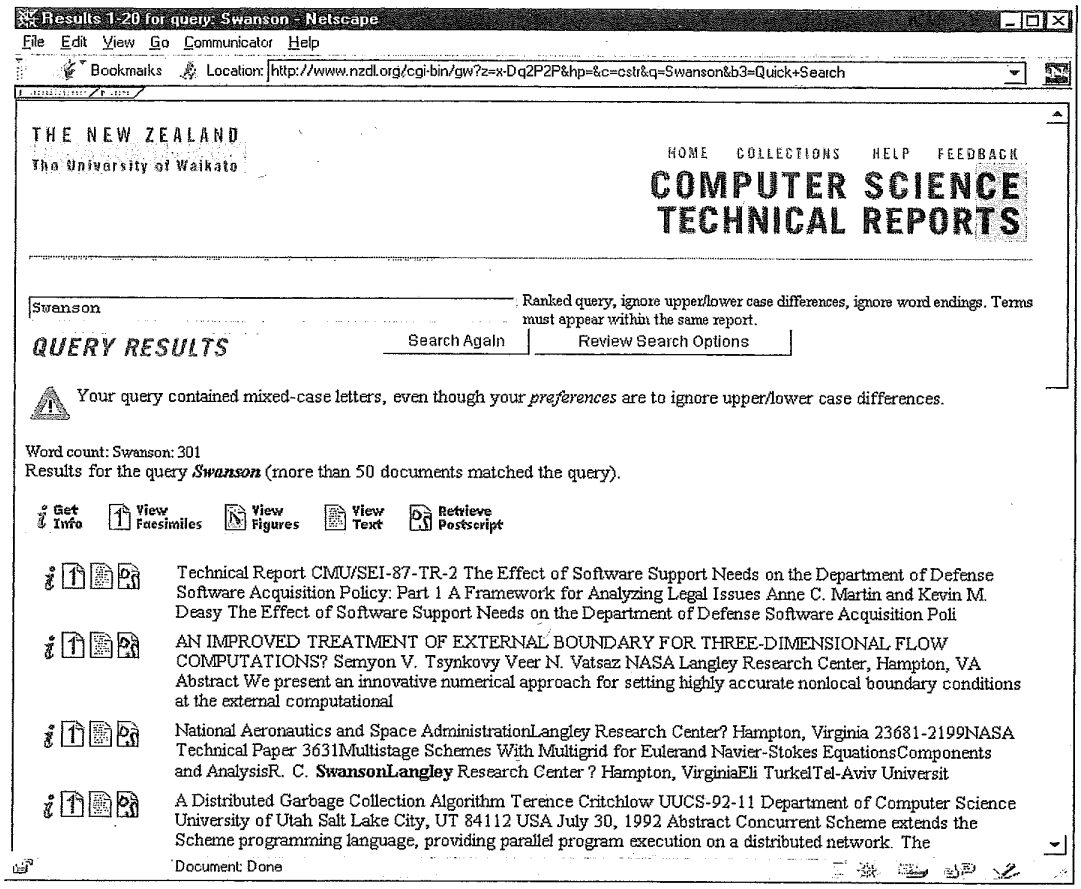


Figure 10.14 An example of a ranked list of titles and other document surrogate information [824].

KWIC

A facility related to highlighting is the *keyword-in-context* (KWIC) document surrogate. Sentence fragments, full sentences, or groups of sentences that contain query terms are extracted from the full text and presented for viewing along with other kinds of surrogate information (such as document title and abstract). Note that a KWIC listing is different than an abstract. An abstract summarizes the main topics of the document but might not contain references to the terms within the query. A KWIC extract shows sentences that summarize the ways the query terms are used within the document. This display can show not only which subsets of query terms occur in the retrieved documents, but also the context they appear in with respect to one another.

Tradeoff decisions must be made between how many lines of text to show and which lines to display. It is not known which contexts are best selected for viewing but results from text summarization research suggest that the best fragments to show are those that appear near the beginning of the document and that contain the largest subset of query terms [464]. If users have specified which

terms are more important than others, then those fragments containing important terms should be shown before those that contain only less important terms. However, to help retain coherence of the excerpts, selected sentences should be shown in order of their occurrence in the original document, independent of how many search terms they contain.

The KWIC facility is usually not shown in Web search result display, most likely because the system must have a copy of the original document available from which to extract the sentences containing the search terms. Web search engines typically only retain the index without term position information. Systems that index individual Web sites can show KWIC information in the document list display.

TileBars

A more compact form of query term hit display is made available through the TileBars interface. The user enters a query in a faceted format, with one topic per line. After the system retrieves documents (using a quorum or statistical ranking algorithm), a graphical bar is displayed next to the title of each document showing the degree of match for each facet. TileBars thus illustrate at a glance which passages in each article contain which topics – and moreover, how frequently each topic is mentioned (darker squares represent more frequent matches).

Each document is represented by a rectangular bar. Figure 10.15 shows an example. The bar is subdivided into rows that correspond to the query facets. The top row of each TileBar corresponds to 'osteoporosis,' the second row to 'prevention,' and the third row to 'research.' The bar is also subdivided into columns, where each column refers to a passage within the document. Hits that overlap within the same passage are more likely to indicate a relevant document than hits that are widely dispersed throughout the document [356]. The patterns are meant to indicate whether terms from a facet occur as a main topic throughout the document, as a subtopic, or are just mentioned in passing.

The darkness of each square corresponds to the number of times the query occurs in that segment of text; the darker the square the greater the number of hits. White indicates no hits on the query term. Thus, the user can quickly see if some subset of the terms overlap in the same segment of the document. (The segments for this version of the interface are fixed blocks of 100 tokens each.)

The first document can be seen to have considerable overlap among the topics of interest towards the middle, but not at the beginning or the end (the actual end is cut off). Thus it most likely discusses topics in addition to research into osteoporosis. The second through fourth documents, which are considerably shorter, also have overlap among all terms of interest, and so are also probably of interest to the user. (The titles help to verify this.) The next three documents are all long, and from the TileBars we can tell they discuss research and prevention, but do not even touch on osteoporosis, and so probably are not of interest.

Because the TileBars interface allows the user to specify the query in terms

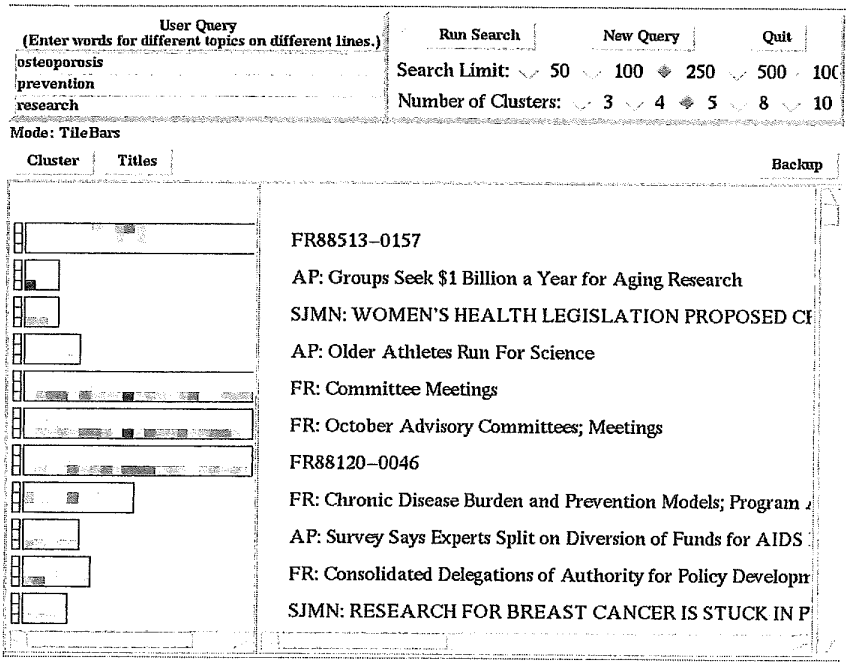


Figure 10.15 An example of the TileBars retrieval results visualization [355].

of facets, where the terms for each facet are listed on an entry line, a color can be assigned to each facet. When the user displays a document with query term hits, the user can quickly ascertain what proportion of search topics appear in a passage based only on how many different highlight colors are visible. Most systems that use highlighting use only a single color to bring attention to all of the search terms.

It would be difficult for users to specify in advance which patterns of term hits they are interested in. Instead, TileBars allows users to scan graphic representations and recognize which documents are and are not of interest. It may be the case that TileBars may be most useful for helping users discard misleadingly interesting documents, but only preliminary studies have been conducted to date. Passages can correspond to paragraphs or sections, fixed sized units of arbitrary length, or to automatically determined multiparagraph segments [355].

SeeSoft

The SeeSoft visualization [232] represents text in a manner resembling columns of newspaper text, with one 'line' of text on each horizontal line of the strip. (See Figure 10.16.) The representation is compact and aesthetically pleasing. Graphics are used to abstract away the details, providing an overview showing the amount and shape of the text. Color highlighting is used to pick out various attributes, such as where a particular word appears in the text. Details of a smaller portion of the display can be viewed via a pop-up window; the overview

shows more of the text but in less detail.

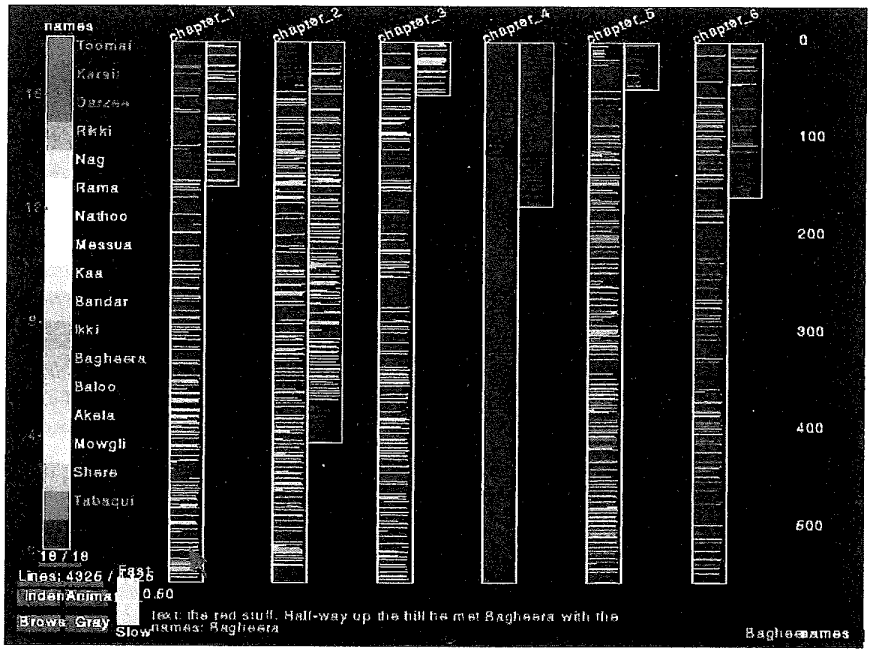


Figure 10.16 An example of the SeeSoft visualization for showing locations of characters within a text [232].

SeeSoft was originally designed for software development, in which a line of text is a meaningful unit of information. (Programmers tend to place each individual programming statement on one line of text.) Thus SeeSoft shows attributes relevant to the programming domain, such as which lines of code were modified by which programmer, and how often particular lines have been modified, and how many days have elapsed since the lines were last modified. The SeeSoft developers then experimented with applying this idea to the display of text, although this has not been integrated into an information access system. Color highlighting is used to show which characters appear where in a book of fiction, and which passages of the Bible contain references to particular people and items. Note the use of the abstraction of an entire line to stand for a single word such as a character's name (even though though this might obscure a tightly interwoven conversation between two characters).

10.6.3 Query Term Hits Between Documents

Other visualization ideas have been developed to show a different kind of information about the relationship between query terms and retrieved documents. Rather than showing how query terms appear within individual documents, as is done in KWIC interfaces and TileBars, these systems display an overview or summary of the retrieved documents according to which subset of query terms they contain. The following subsections describe variations on this idea.

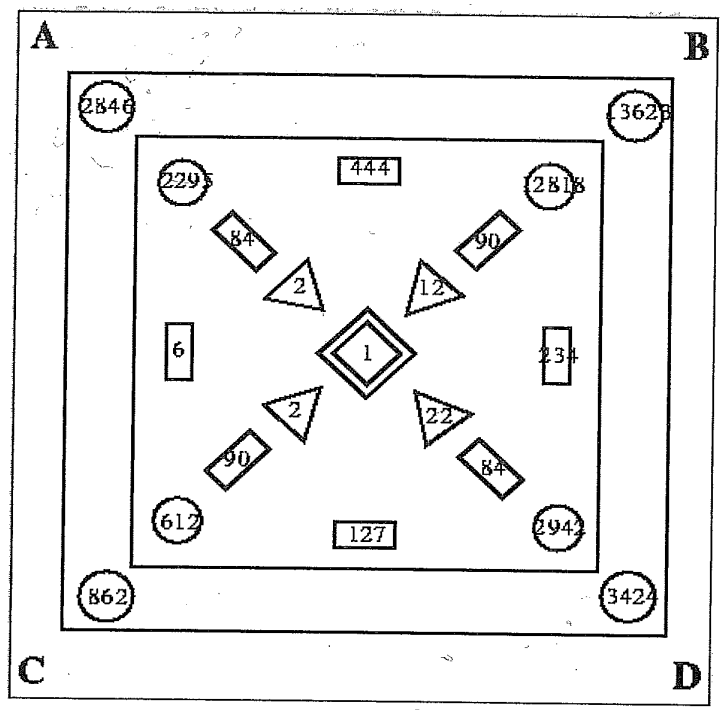


Figure 10.17 A sketch of the InfoCrystal retrieval results display [738].

InfoCrystal

The InfoCrystal shows how many documents contain each subset of query terms [738]. This relieves the user from the need to specify Boolean ANDs and ORs in their query, while still showing which combinations of terms actually appear in documents that were ordered by a statistical ranking (although beyond four terms the interface becomes difficult to understand). The InfoCrystal allows visualization of all possible relations among N user-specified 'concepts' (or Boolean keywords). The InfoCrystal displays, in a clever extension of the Venn diagram paradigm, the number of documents retrieved that have each possible subset of the N concepts. Figure 10.17 shows a sketch of what the InfoCrystal might display as the result of a query against four keywords or Boolean phrases, labeled A, B, C, and D. The diamond in the center indicates that one document was discovered that contains all four keywords. The triangle marked with '12' indicates that 12 documents were found containing attributes A, B, and D, and so on.

The InfoCrystal does not show proximity among the terms within the documents, nor their relative frequency. So a document that contains dozens of hits on 'volcano' and 'lava' and one hit on 'Mars' will be grouped with documents that contain mainly hits on 'Mars' but just one mention each of 'volcano' and 'lava.'

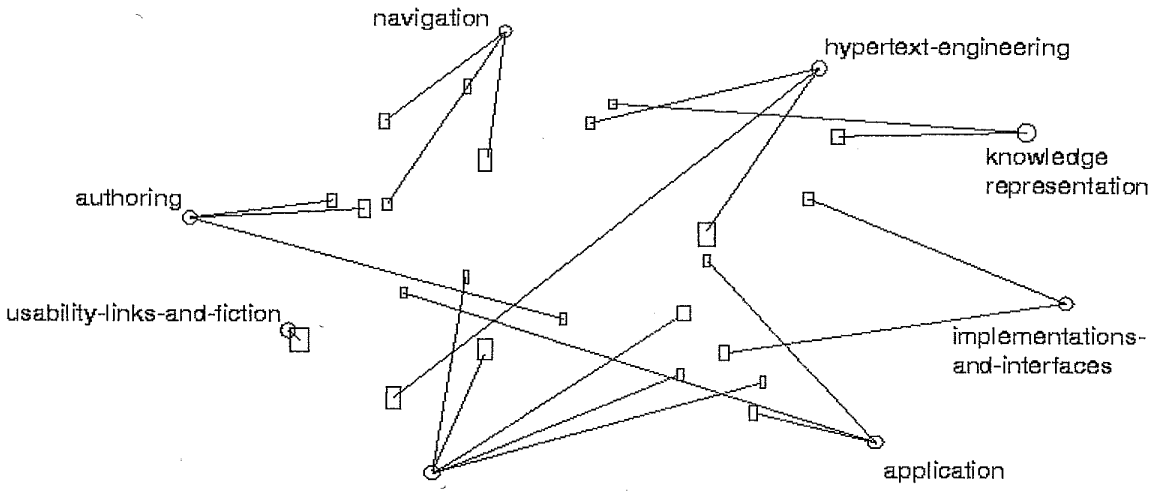


Figure 10.18 An example of the VIBE retrieval results display [452].

VIBE and Lyberworld

Graphical presentations that operate on similar principles are VIBE [452] and Lyberworld [363]. In these displays, query terms are placed in an abstract graphical space. After the search, icons are created that indicate how many documents contain each subset of query terms. The subset status of each group of documents is indicated by the placement of the icon. For example, in VIBE a set of documents that contain three out of five query terms are shown on an axis connecting these three terms, at a point midway between the representations of the three query terms in question. (See Figure 10.18.) Lyberworld presents a 3D version of this idea.

Lattices

Several researchers have employed a graphical depiction of a mathematical lattice for the purposes of query formulation, where the query consists of a set of constraints on a hierarchy of categories (actually, semantic attributes in these systems) [631, 147]. This is one solution to the problem of displaying documents in terms of multiple attributes; a document containing terms A, B, C, and D could be placed at a point in the lattice with these four categories as parents. However, if such a representation were to be applied to retrieval results instead of query formulation, the lattice layout would in most cases be too complex to allow for readability.

None of the displays discussed in this subsection have been evaluated for effectiveness at improving query specification or understanding of retrieval results, but they are intriguing ideas and perhaps are useful in conjunction with other displays.

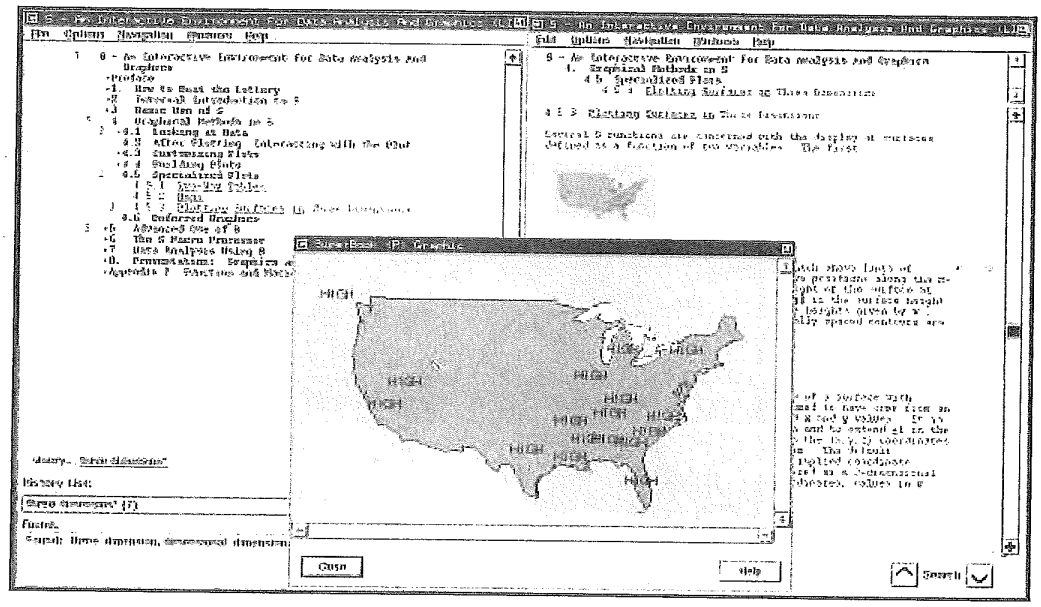


Figure 10.19 The SuperBook interface for showing retrieval results on a large manual in context [481].

10.6.4 SuperBook: Context via Table of Contents

The SuperBook system [481, 229, 230] makes use of the structure of a large document to display query term hits in context. The table of contents (TOC) for a book or manual are shown in a hierarchy on the left-hand side of the display, and full text of a page or section is shown on the right-hand side. The user can manipulate the table of contents to expand or contract the view of sections and subsections. A focus-plus-context mechanism is used to expand the viewing area of the sections currently being looked at and compress the remaining sections. When the user moves the cursor to another part of the TOC, the display changes dynamically, making the new focus larger and shrinking down the previously observed sections.

After the user specifies a query on the book, the search results are shown in the context of the table of contents hierarchy. (See Figure 10.19.) Those sections that contain search hits are made larger and the others are compressed. The query terms that appear in chapter or section names are highlighted in reverse video. When the user selects a page from the table of contents view, the page itself is displayed on the right-hand side and the query terms within the page are highlighted in reverse video.

The SuperBook designers created innovative techniques for evaluating its special features. Subjects were compared using this system against using paper documentation and against a more standard online information access system. Subjects were also compared on different kinds of carefully selected tasks: browsing topics of interest, citation searching, searching to answer questions, and searching and browsing to write summary essays. For most of the tasks

SuperBook subjects were faster and more accurate or equivalent in speed and accuracy to a standard system. When differences arose between SuperBook and the standard system, the investigators examined the logs carefully and hypothesized plausible explanations. After the initial studies, they modified SuperBook according to these hypotheses and usually saw improvements as a result [481].

The user studies on the improved system showed that users were faster and more accurate at answering questions in which some of the relevant terms were within the section titles themselves, but they were also faster and more accurate at answering questions in which the query terms fell within the full text of the document only, as compared both to a paper manual and to an interface that did not provide such contextualizing information. SuperBook was not faster than paper when the query terms did not appear in the document text or the table of contents. This and other evidence from the SuperBook studies suggests that query term highlighting is at least partially responsible for improvements seen in the system.

10.6.5 Categories for Results Set Context

In section 10.4 we saw the use of category or directory information for providing overviews of text collection content. Category metadata can also be used to place the results of a query in context.

For example, the original formulation of SuperBook allowed navigation within a highly structured document, a computer manual. The CORE project extended the main idea to a collection of over 1000 full-text chemistry articles. A study of this representation demonstrated its superiority to a standard search system on a variety of task types [228]. Since a table of contents is not available for this collection, context is provided by placing documents within a category hierarchy containing terms relevant to chemistry. Documents assigned a category are listed when that category is selected for more detailed viewing, and the categories themselves are organized into a hierarchy, thus providing a hierarchical view on the collection.

Another approach to using predefined categories to provide context for retrieval results is demonstrated by the DynaCat system [650]. The DynaCat system organizes retrieved documents according to which types of categories, selected from the large MeSH taxonomy, are known in advance to be important for a given query type. DynaCat begins with a set of query types known to be useful for a given user population and collection. One query type can encompass many different queries. For example, the query type 'Treatment-Adverse Effects' covers queries such as 'What are the complications of a mastectomy?' as well as 'What are the side-effects of aspirin?' Documents are organized according to a set of criteria associated with each query type. These criteria specify which types of categories that are acceptable to use for organizing the documents and consequently, which categories should be omitted from the display. Once categories have been assigned to the retrieved documents, a hierarchy is formed based on where the categories exist within MeSH. The algorithm selects only a subset of the category

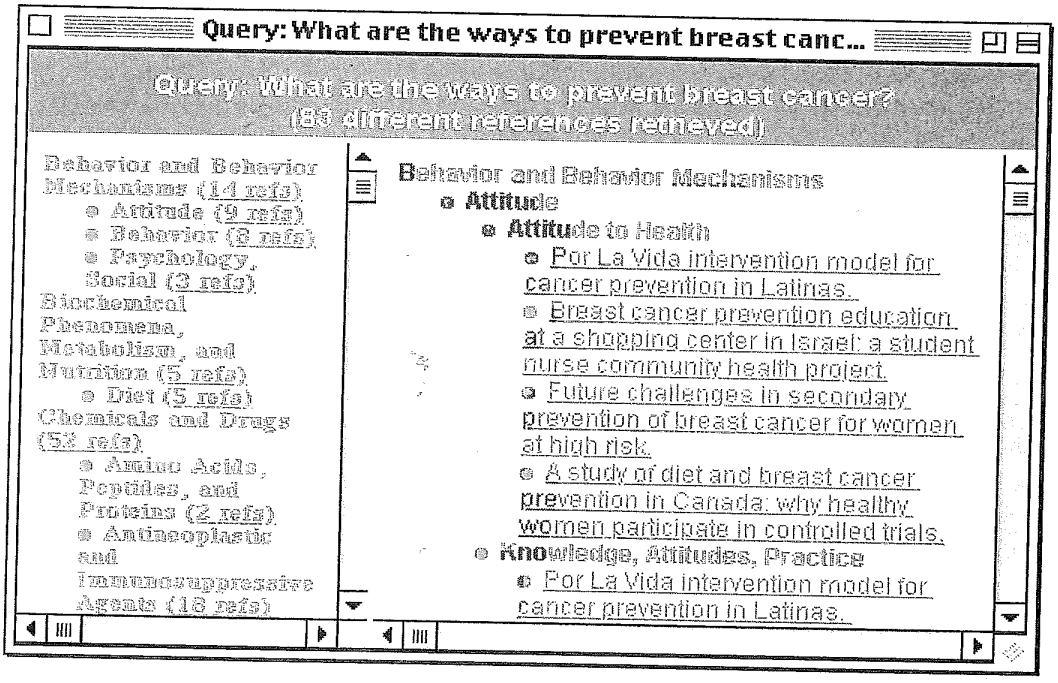


Figure 10.20 The DynaCat interface for viewing category labels that correspond to query types [650].

labels that might be assigned to the document to be used in the organization.

Figure 10.20 shows the results for a query on breast cancer prevention. The interface is tiled into three windows. The top window displays the user's query and the number of documents found. The left window shows the categories in the first two levels of the hierarchy, providing a table of contents view of the organization of search results. The right pane displays all the categories in the hierarchy and the titles of the documents that belong in those categories.

An obstacle to using category labels to organize retrieval results is the requirement of precompiled knowledge about which categories are of interest for a particular user or a particular query type. The SONIA system [692] circumvents this problem by using a combination of unsupervised and supervised methods to organize a set of documents. The unsupervised method (document clustering similar to Scatter/Gather) imposes an initial organization on a user's personal information collection or on a set of documents retrieved as the result of a query. The user can then invoke a direct manipulation interface to make adjustments to this initial clustering, causing it to align more closely with their preferences (because unsupervised methods do not usually produce an organization that corresponds to a human-derived category structure [357]). The resulting organization is then used to train a supervised text categorization algorithm which automatically classifies any new documents that are added to the collection. As the collection grows it can be periodically reorganized by rerunning the clustering algorithm and redoing the manual adjustments.

10.6.6 Using Hyperlinks to Organize Retrieval Results

Although the SuperBook authors describe it as a hypertext system, it is actually better thought of as a means of showing search results in the context of a structure that users can understand and view all at once. The hypertext component was not analyzed separately to assess its importance, but it usually is not mentioned by the authors when describing what is successful about their design. In fact, it seems to be responsible for one of the main problems seen with the revised version of the system — that users tend to wander off (often unintentionally) from the pages they are reading, thus causing the time spent on a given topic to be longer for SuperBook in some cases. (Using completion time to evaluate users on browsing tasks can be problematic, however, since by definition browsing is a casual, unhurried process [804].)

This wandering may occur in part because SuperBook uses a non-standard kind of hypertext, in which any word is automatically linked to occurrences of the same word in other parts of the document. This has not turned out to be how hypertext links are created in practice. Today, hyperlinked help systems and hyperlinks on the Web make much more discriminating use of hyperlink connections (in part since they are usually generated by an author rather than automatically). These links tend to be labeled in a somewhat meaningful manner by their surrounding context. Back-of-the-book indexes often do not contain listings of every occurrence of a word, but rather to the more important uses or the beginnings of series of uses. Automated hypertext linking should perhaps be based on similar principles. Additionally, at least one study showed that users formed better mental models of a small hypertext system that was organized hierarchically than one that allowed more flexible access [226]. Problems relating to navigation of hypertext structure have long been suspected and investigated in the hypertext literature [181, 551, 440, 334].

More recent work has made better use of hyperlink information for providing context for retrieval results. Some of this work is described below.

Cha-Cha: SuperBook on the Web

The Cha-Cha intranet search system [164] extends the SuperBook idea to a large heterogeneous Web site such as might be found in an organization's intranet. Figure 10.21 shows an example. This system differs from SuperBook in several ways. On most Web sites there is no existing real table of contents or category structure, and an intranet like those found at large universities or large corporations is usually not organized by one central unit. Cha-Cha uses link structure present within the site to create what is intended to be a meaningful organization on top of the underlying chaos. After the user issues a query, the shortest paths from the root page to each of the search hits are recorded and a subset of these are selected to be shown as a hierarchy, so that each hit is shown only once. (Users can begin with a query, rather than with a table of contents view.) If a user does not know to use the term 'health center' but instead queries on 'medical center,' if 'medical' appears as a term in a document within

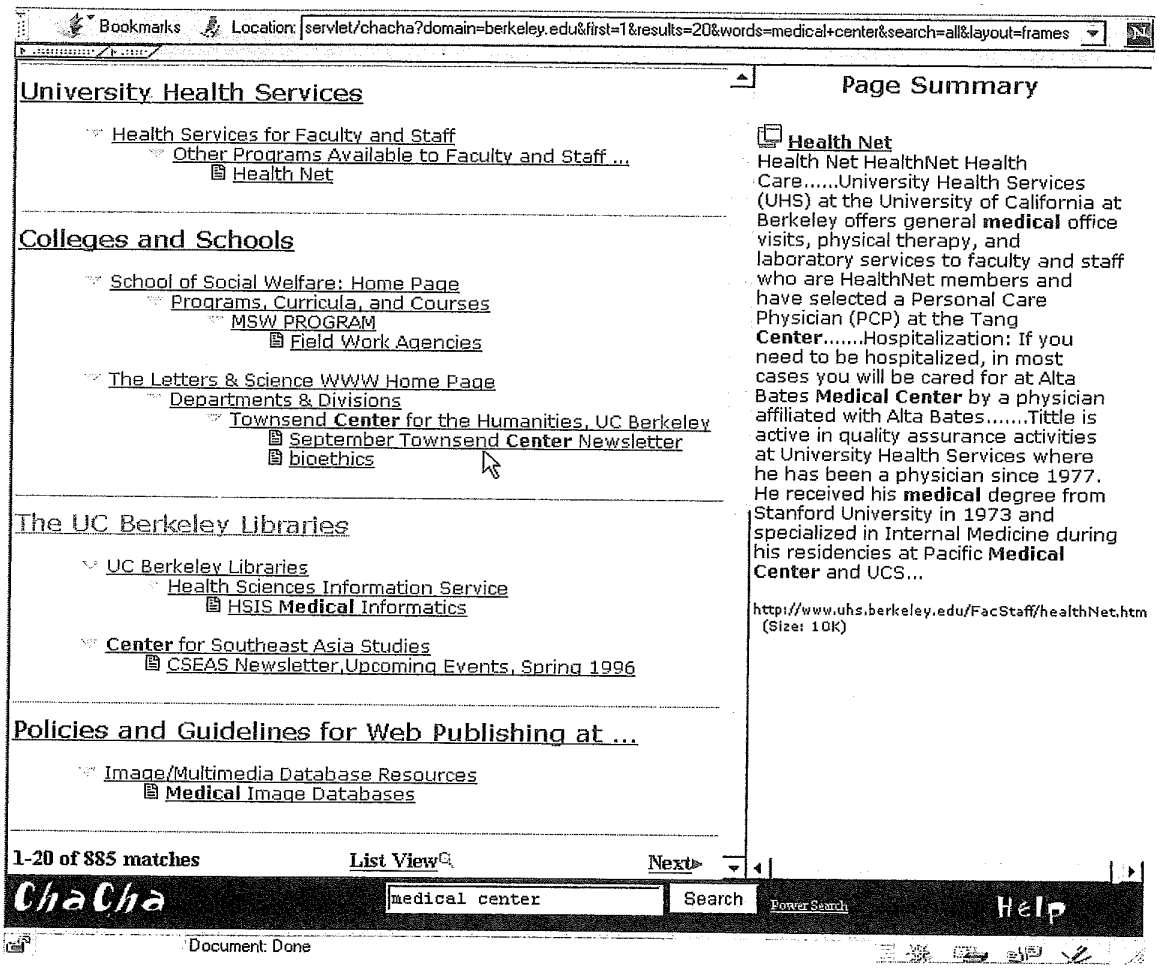


Figure 10.21 The Cha-Cha interface for showing Web intranet search results in context displaying results on the query 'medical centre'[164].

the health center part of the Web, the home page (or starting point) of this center will be presented as well as the more specific hits. Users can then either query or navigate within a subset of sites if they wish. The organization produced by this simple method is surprisingly comprehensible on the UC Berkeley site. It seems especially useful for providing the information about the sources (the Web server) associated with the search hits, whose titles are often cryptic.

The AMIT system [826] also applies the basic ideas behind SuperBook to the Web, but focuses on a single-topic Web site, which is likely to have a more reasonable topic structure than a complex intranet. The link structure of the Web site is used as contextualizing information but all of the paths to a given document are shown and focus-plus-context is used to emphasize subsets of the document space. The WebTOC system [585] is similar to AMIT but focuses on showing the structure and number of documents within each Web subhierarchy, and is not tightly coupled with search.

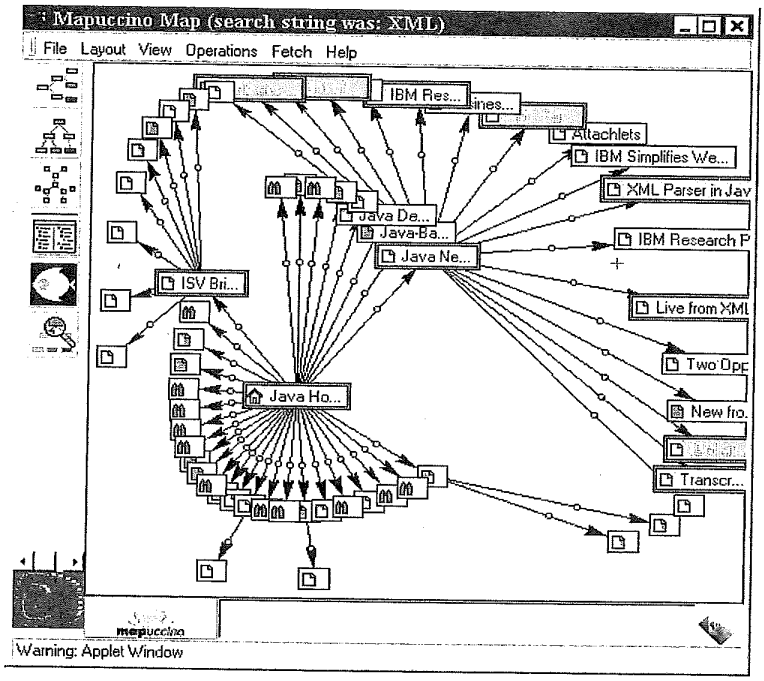


Figure 10.22 Example of a Web subset visualized by Mapuccino (courtesy of M. Jacovi, B. Shaul and Y. Maarek).

Mapuccino: Graphical Depiction of Link Structure

The Mapuccino system (formerly WebCutter) [527] allows the user to issue a query on a particular Web site. The system crawls the site in real-time, checking each encountered page for relevance to the query. When a relevant page is found, the weights on that page's outlinks are increased. Thus, the search is based partly on an assumption that relevant pages will occur near one another in the Web site. The subset of the Web site that has been crawled is depicted graphically in a nodes-and-links view (see Figure 10.22). This kind of display does not provide the user with information about what the *contents* of the pages are, but rather only shows their link structure. Other researchers have also investigated spreading activation among hypertext links as a way to guide an information retrieval system, e.g., [278, 555].

10.6.7 Tables

Tabular display is another approach for showing relationships among retrieval documents. The Envision system [273] allows the user to organize results according to metadata such as author or date along the X and Y-axes, and uses graphics to show values for attributes associated with retrieved documents within each cell (see Figure 10.23). Color, shape, and size of an iconic representation of a document are used to show the computed relevance, the type of document, or

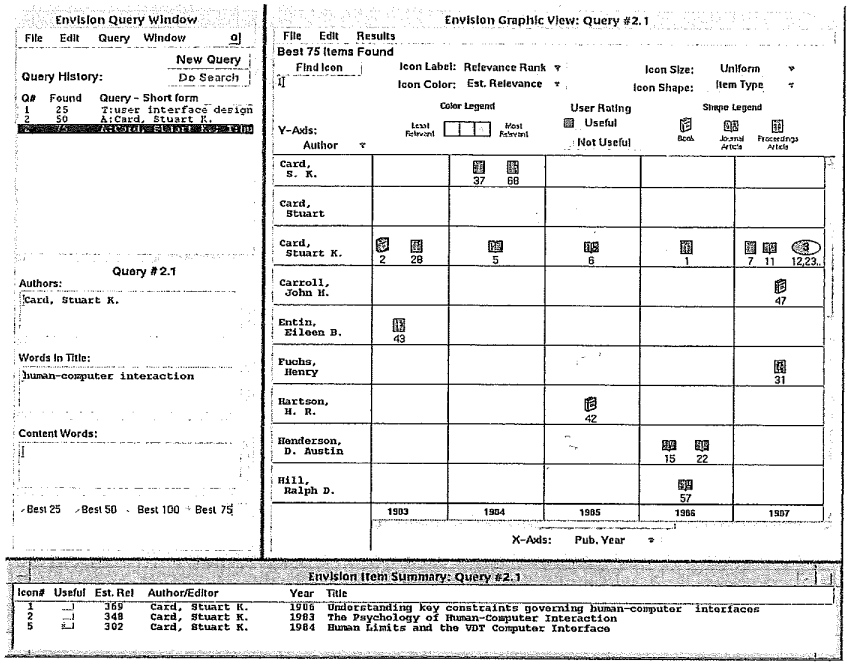


Figure 10.23 The Envision tabular display for graphically organizing retrieved documents [270].

other attributes. Clicking on an icon brings up more information about the document in another window. Like the WebCutter system, this view provides few cues about how the documents are related to one another in terms of their content or meaning. The SenseMaker system also allows users to group documents into different views via a table-like display [51], including a Scatter/Gather [203] style view. Although tables are appealing, they cannot show the intersections of many different attributes; rather they are better for pairwise comparisons. Another problem with tables for display of textual information is that very little information can be fitted on a screen at a time, making comparisons difficult.

The Table Lens [666] is an innovative interface for viewing and interactively reorganizing very large tables of information (see Figure 10.24). It uses focus-plus-context to fit hundreds of rows of information in a space occupied by at most two dozen rows in standard spreadsheets. And because it allows for rapid reorganization via sorting of columns, users can quickly switch from a view focused around one kind of metadata to another. For example, first sorting documents by rank and then by author name can show the relative ranks of different articles by the same author. A re-sort by date can show patterns in relevance scores with respect to date of publication. This rapid re-sorting capability helps circumvent the problems associated with the fact that tables cannot show many simultaneous intersections.

Another variation on the table theme is that seen in the Perspective Wall [530] in which a focus-plus-context display is used to center information currently

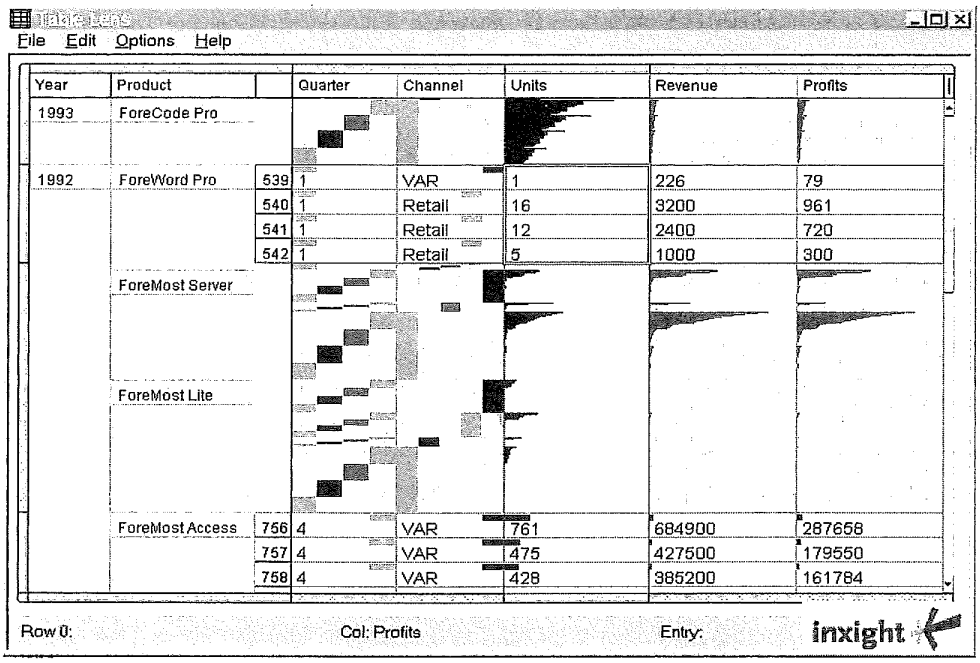


Figure 10.24 The TableLens visualization [666].

of interest in the middle of the display, compressing less important information into the periphery on the sides of the wall. The idea is to show in detail the currently most important information while at the same time retaining the context of the rest of the information. For example, if viewing documents in chronological order, the user can easily tell if they are currently looking at documents in the beginning, middle, or end of the time range.

These interfaces have not been applied to information access tasks. The problem with such displays when applied to text is that they require an attribute that can be shown according to an underlying order, such as date. Unfortunately, information useful for organizing text content, such as topic labels, does not have an inherent meaningful order. Alphabetical order is useful for looking up individual items, but not for seeing patterns across items according to adjacency, as in the case for ordered data types like dates and size.

10.7 Using Relevance Judgements

An important part of the information access process is query reformulation, and a proven effective technique for query reformulation is *relevance feedback*. In its original form, relevance feedback refers to an interaction cycle in which the user selects a small set of documents that appear to be relevant to the query, and the system then uses features derived from these selected relevant documents to revise the original query. This revised query is then executed and a new set of documents is returned. Documents from the original set can appear in the new results

list, although they are likely to appear in a different rank order. Relevance feedback in its original form has been shown to be an effective mechanism for improving retrieval results in a variety of studies and settings [702, 343, 127]. In recent years the scope of ideas that can be classified under this term has widened greatly.

Relevance feedback introduces important design choices, including which operations should be performed automatically by the system and which should be user initiated and controlled. Bates discusses this issue in detail [66], asserting that despite the emphasis in modern systems to try to automate the entire process, an intermediate approach in which the system helps automate search at a *strategic* level is preferable. Bates suggests an analogy of an automatic camera versus one with adjustable lenses and shutter speeds. On many occasions, a quick, easy method that requires little training or thought is appropriate. At other times the user needs more control over the operation of the machinery, while still not wanting to know about the low level details of its operation.

A related idea is that, for any interface, control should be described in terms of the task being done, not in terms of how the machine can be made to accomplish the task [607]. Continuing the camera analogy, the user should be able to control the mood created by the photograph, rather than the adjustment of the lens. In information access systems, control should be over the kind of information returned, not over which terms are used to modify the query. Unfortunately it is often quite difficult to build interfaces to complex systems that behave in this manner.

10.7.1 Interfaces for Standard Relevance Feedback

A standard interface for relevance feedback consists of a list of titles with checkboxes beside the titles that allow the user to mark relevant documents. This can imply either that unmarked documents are not relevant or that no opinion has been made about unmarked documents, depending on the system. Another option is to provide a choice among several checkboxes indicating relevant or not relevant (with no selection implying no opinion). In some cases users are allowed to indicate a value on a relevance scale [73]. Standard relevance feedback algorithms usually do not perform better given negative relevance judgement evidence [225], but machine learning algorithms can take advantage of negative feedback [629, 460].

After the user has made a set of relevance judgements and issued a search command, the system can either automatically reweight the query and re-execute the search, or generate a list of terms for the user to select from in order to augment the original query. (See Figure 10.25, taken from [448].) Systems usually do not suggest terms to remove from the query.

After the query is re-executed, a new list of titles is shown. It can be helpful to retain an indicator such as a marked checkbox beside the documents that the user has already judged. A difficult design decision concerns whether or not to show documents that the user has already viewed towards the top of the ranked list [1]. Repeatedly showing the same set of documents at the top may inconvenience a user who is trying to create a large set of relevant documents,

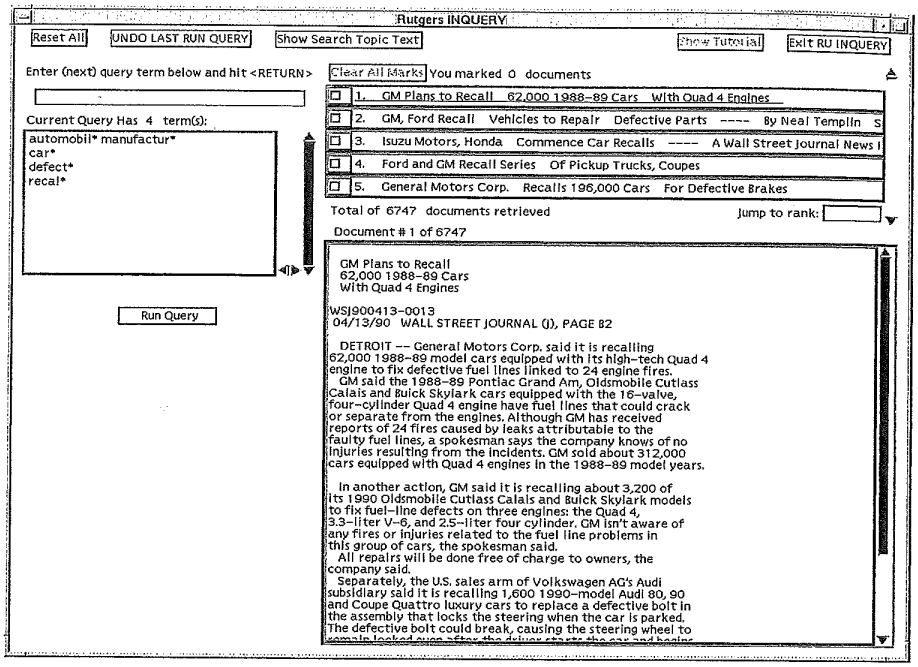


Figure 10.25 An example of an interface for relevance feedback [448].

but at the same time, this can serve as feedback indicating that the revised query does not downgrade the ranking of those documents that have been found especially important. One solution is to retain a separate window that shows the rankings of only the documents that have not been retrieved or ranked highly previously. Another solution is to use smaller fonts or gray-out color for the titles of documents already seen.

Creating multiple relevance judgements is an effortful task, and the notion of relevance feedback is unfamiliar to most users. To circumvent these problems, Web-based search engines have adopted the terminology of 'more like this' as a simpler way to indicate that the user is requesting documents similar to the selected one. This 'one-click' interaction method is simpler than standard relevance feedback dialog which requires users to rate a small number of documents and then request a reranking. Unfortunately, in most cases relevance feedback requires many relevance judgements in order to work well. To partly alleviate this problem, Aalbersberg [1] proposes incremental relevance feedback which works well given only one relevant document at a time and thus can be used to hide the two-step procedure from the user.

10.7.2 Studies of User Interaction with Relevance Feedback Systems

Standard relevance feedback assumes the user is involved in the interaction by specifying the relevant documents. In some interfaces users are also able to

select which terms to add to the query. However, most ranking and reweighting algorithms are difficult to understand or predict (even for the creators of the algorithms!) and so it might be the case that users have difficulties controlling a relevance feedback system explicitly.

A recent study was conducted to investigate directly to what degree user control of the feedback process is beneficial. Koenemann and Belkin [448] measured the benefits of letting users 'under the hood' during relevance feedback. They tested four cases using the Inquery system [772]:

- **Control** No relevance feedback; the subjects could only reformulate the query by hand.
- **Opaque** The subjects simply selected relevant documents and saw the revised rankings.
- **Transparent** The subjects could see how the system reformulated the queries (that is, see which terms were added — the system did not reweight the subjects' query terms) and the revised rankings.
- **Penetrable** The system is stopped midway through the reranking process. The subjects are shown the terms that the system would have used for opaque and transparent query reformulation. The subjects then select which, if any, of the new terms to add to the query. The system then presents the revised rankings.

The 64 subjects were much more effective (measuring precision at a cut-off of top 5, top 10, top 30, and top 100 documents) with relevance feedback than without it. The penetrable group performed significantly better than the control, with the opaque and transparent performances falling between the two in effectiveness. Search times did not differ significantly among the conditions, but there were significant differences in the number of feedback iterations. The subjects in the penetrable group required significantly fewer iterations to achieve better queries (an average of 5.8 cycles in the penetrable group, 8.2 cycles in the control group, 7.7 cycles in the opaque group, and surprisingly, the transparent group required more cycles, 8.8 on average). The average number of documents marked relevant ranged between 11 and 14 for the three conditions. All subjects preferred relevance feedback over the baseline system, and several remarked that they preferred the 'lazy' approach of selecting suggested terms over having to think up their own.

An observational study on a TTY-based version of an online catalog system [338] also found that users performed better using a relevance feedback mechanism that allowed manual selection of terms. However, a later observational study did not find overall success with this form of relevance feedback [337]. The authors attribute these results to a poor design of a new graphical interface. These results may also be due to the fact that users often selected only one relevant document before performing the feedback operation, although they were using a system optimized from multiple document selection.

10.7.3 Fetching Relevant Information in the Background

Standard relevance feedback is predicated on the goal of improving an ad hoc query or building a profile for a routing query. More recently researchers have begun developing systems that monitor users' progress and behavior over long interaction periods in an attempt to predict which documents or actions the user is likely to want in future. These systems are called semi-automated *assistants* or recommender 'agents,' and often make use of machine learning techniques [565]. Some of these systems require explicit user input in the form of a goal statement [406] or relevance judgements [629], while others quietly record users' actions and try to make inferences based on these actions.

A system developed by Kozierok and Maes [460, 536] makes predictions about how users will handle email messages (what order to read them in, where to file them) and how users will schedule meetings in a calendar manager application. The system 'looks over the shoulder' of the users, recording every relevant action into a database. After enough data has been accumulated, the system uses a nearest-neighbors method [743] to predict a user's action based on the similarity of the current situation to situations already encountered. For example, if the user almost always saves email messages from a particular person into a particular file, the system can offer to automate this action the next time a message from that person arrives [536]. This system integrates learning from both implicit and explicit user feedback. If a user ignores the system's suggestion, the system treats this as negative feedback, and accordingly adds the overriding action to the action database. After certain types of incorrect predictions, the system asks the user questions that allow it to adjust the weight of the feature that caused the error. Finally, the user can explicitly train the system by presenting it with hypothetical examples of input-action pairs.

Another system, Syskill and Webert [629], attempts to learn a user profile based on explicit relevance judgements of pages explored while browsing the Web. In a sense this is akin to standard relevance feedback, except the user judgements are retained across sessions and the interaction model differs: as the user browses a new Web page, the links on the page are automatically annotated as to whether or not they should be relevant to the user's interest.

A related system is Letizia [518], whose goal is to bring to the user's attention a percentage of the available next moves that are most likely to be of interest, given the user's earlier actions. Upon request, Letizia provides recommendations for further action on the user's part, usually in the form of suggestions of links to follow when the user is unsure what to do next. The system monitors the user's behavior while navigating and reading Web pages, and concurrently evaluates the links reachable from the current page. The system uses only implicit feedback. Thus, saving a page as a bookmark is taken as strong positive evidence for the terms in the corresponding Web page. Links skipped are taken as negative support for the information reachable from the link. Selected links can indicate positive or negative evidence, depending on how much time the user spends on the resulting page and whether or not the decision to leave a page quickly is later reversed. Additionally, the evidence for user interest remains persistent across

browsing sessions. Thus, a user who often reads kayaking pages is at another time reading the home page of a professional contact and may be alerted to the fact that the colleague's personal interests page contains a link to a shared hobby. The system uses a best-first search strategy and heuristics to determine which pages to recommend most strongly.

A more user-directed approach to prefetching potentially relevant information is seen in the Butterfly system [531]. This interface helps the user follow a series of citation links from a given reference, an important information seeking strategy [66]. The system automatically examines the document the user is currently reading and prefetches the bibliographic citations it refers to. It also retrieves lists of articles that cite the focus document. The underlying assumption is that the services from which the citations are requested do not respond immediately. Rather than making the user wait during the delay associated with each request, the system handles many requests in parallel and the interface uses graphics and animations to show the incrementally growing list of available citations. The system does not try to be clever about which cites to bring first; rather the user can watch the 'organically' growing visualization of the document and its citations, and based on what looks relevant, direct the system as to which parts of the citation space to spend more time on.

10.7.4 Group Relevance Judgements

Recently there has been much interest in using relevance judgements from a large number of different users to rate or rank information of general interest [672]. Some variations of this *social recommendation* approach use only similarity among relevance judgements by people with similar tastes, ignoring the representation of the information being judged altogether. This has been found highly effective for rating information in which taste plays a major role, such as movie and music recommendations [720]. More recent work has combined group relevance judgements with content information [64].

10.7.5 Pseudo-Relevance Feedback

At the far end of the system versus user feedback spectrum is what is informally known as pseudo-relevance feedback. In this method, rather than relying on the user to choose the top k relevant documents, the system simply assumes that its top-ranked documents are relevant, and uses these documents to augment the query with a relevance feedback ranking algorithm. This procedure has been found to be highly effective in some settings [760, 465, 12], most likely those in which the original query statement is long and precise. An intriguing extension to this idea is to use the output of clustering of retrieval results as the input to a relevance feedback mechanism, either by having the user or the system select the cluster to be used [359], but this idea has not yet been evaluated.

10.8 Interface Support for the Search Process

The user interface designer must make decisions about how to arrange various kinds of information on the computer screen and how to structure the possible sequences of interactions. This design problem is especially daunting for a complex activity like information access. In this section we discuss design choices surrounding the layout of information within complex information systems, and illustrate the ideas with examples of existing interfaces. We begin with a discussion of very simple search interfaces, those used for string search in 'find' operations, and then progress to multiwindow interfaces and sophisticated workspaces. This is followed by a discussion of the integration of scanning, selecting, and querying within information access interfaces and concludes with interface support for retaining the history of the search process.

10.8.1 Interfaces for String Matching

A common simple search need is that of the 'find' operation, typically run over the contents of a document that is currently being viewed. Usually this function does not produce ranked output, nor allow Boolean combinations of terms; the main operation is a simple string match (without regular expression capabilities). Typically a special purpose search window is created, containing a few simple controls (e.g., case-sensitivity, search forward or backward). The user types the query string into an entry form and string matches are highlighted in the target document (see Figure 10.26).

The next degree of complexity is the 'find' function for searching across small collections, such as the files on a personal computer's hard disk, or the history list of a Web browser. This type of function is also usually implemented as a simple string match. Again, the controls and parameter settings are shown at the top of a special purpose search window and the various options are set via checkboxes and entry forms. The difference from the previous example is that a results list is shown within the search interface itself (see Figure 10.27).

A common problem arises even in these very simple interfaces. An ambiguous state occurs in which the results for an earlier search are shown while the user is entering a new query or modifying the previous one. If the user types in

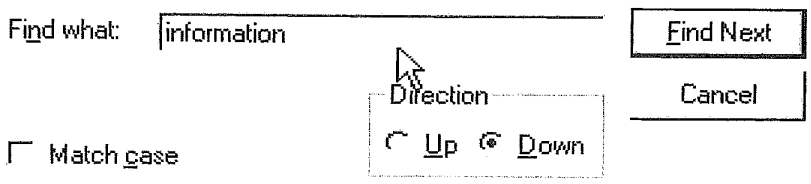


Figure 10.26 An example of a simple interface for string matching, from Netscape Communicator 4.05.

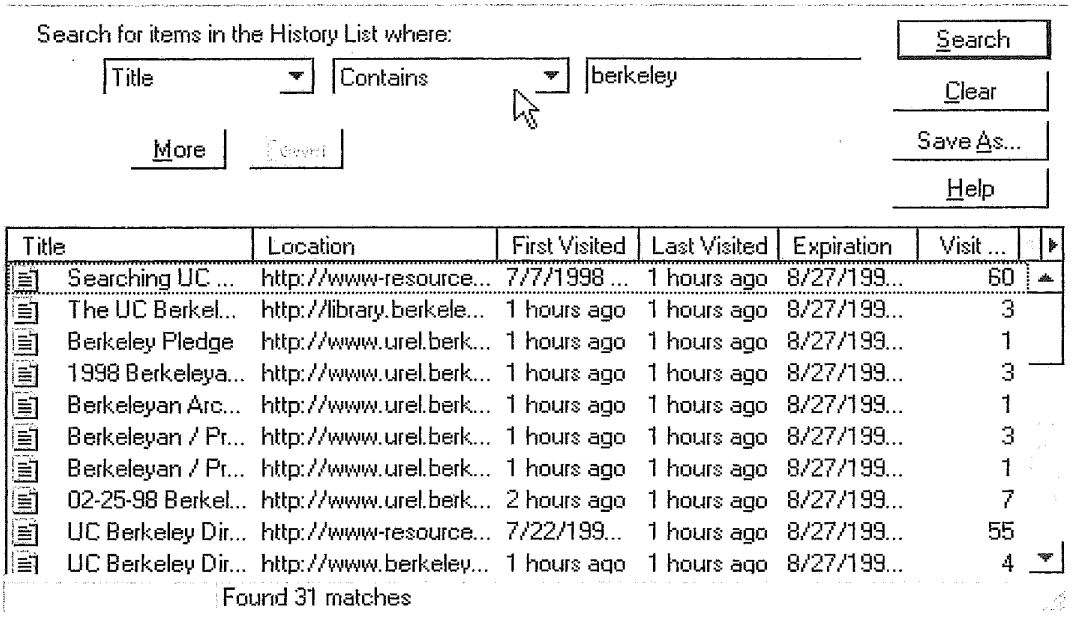


Figure 10.27 An example of an string matching over a list, in this case, a history of recently viewed Web pages, from Netscape Communicator 4.05.

new terms and but then does not activate the search, the interface takes on a potentially misleading state, since a user could erroneously assume that the old search hits shown correspond to the newly typed-in query. One solution for this problem is to clear the results list as soon as the user begins to type in a new query.

However, the user may want to refer to terms shown in the search results to help reformulate the query, or may decide not to issue the new query and instead continue with the previous results. These goals would be hampered by erasing the current result set as soon as the new query is typed. Another solution is to bring up a new window for every new query. However, this requires the user to execute an additional command and can lead to a proliferation of windows. A third, probably more workable solution, is to automatically 'stack' the queries and results lists in a compact format and allow the user to move back and forth among the stacked up prior searches.

Simple interfaces like these can be augmented with functionality that can greatly aid initial query formulation. Spelling errors are a major cause of void result sets. A spell-checking function that suggests alternatives for query terms that have low frequency in the collection might be useful at this stage. Another option is to suggest thesaurus terms associated with the query terms at the time the query terms are entered. Usually these kinds of information are shown after the query is entered and documents have been retrieved, but an alternative is to provide this information as the user enters the query, in a form of query preview.

10.8.2 Window Management

For search tasks more complex than the simple string matching find operations described above, the interface designer must decide how to lay out the various choices and information displays within the interface.

As discussed above, traditional bibliographic search systems use TTY-based command-line interfaces or menus. When the system responds to a command, the new results screen obliterates the contents of the one before it, requiring the user to remember the context. For example, the user can usually see only one level of a subject hierarchy at a time, and must leave the subject view in order to see query view or the document view. The main design choices in such a system are in the command or menu structure, and the order of presentation of the available options.

In modern graphical interfaces, the windowing system can be used to divide functionality into different, simultaneously displayed views [582]. In information access systems, it is often useful to link the information from one window to the information in another, for example, linking documents to their position in a table of contents, as seen in SuperBook. Users can also use the selection to cut and paste information from one window into another, for example, copy a word from a display of thesaurus terms and paste the word into the query specification form.

When arranging information within windows, the designer must choose between a *monolithic* display, in which all the windows are laid out in predefined positions and are all simultaneously viewable, *tiled windows*, and *overlapping windows*. User studies have been conducted comparing these options when applied to various tasks [725, 96]. Usually the results of these studies depend on the domain in which the interface is used, and no clear guidelines have yet emerged for information access interfaces.

The monolithic interface has several advantages. It allows the designer to control the organization of the various options, makes all the information simultaneously viewable, and places the features in familiar positions, making them easier to find. But monolithic interfaces have disadvantages as well. They often work best if occupying the full viewing screen, and the number of views is inherently limited by the amount of room available on the screen (as opposed to overlapping windows which allow display of more information than can fit on the screen at once). Many modern work-intensive applications adopt a monolithic design, but this can hamper the integration of information access with other work processes such as text editing and data analysis. Plaisant *et al.* [644] discuss issues relating to coordinating information across different windows to providing overview plus details.

A problem for any information access interface is an inherent limit in how many kinds of information can be shown at once. Information access systems must always reserve room for a text display area, and this must take up a significant proportion of screen space in order for the text to be legible. A tool within a paint program, for example, can be made quite small while nevertheless remaining recognizable and usable. For legibility reasons, it is difficult to compress many of the information displays needed for an information access system (such

as lists of thesaurus terms, query specifications, and lists of saved titles) in this manner. Good layout, graphics, and font design can improve the situation; for example, Web search results can look radically different depending on spacing, font, and other small touches [580].

Overlapping windows provide flexibility in arrangement, but can quickly lead to a crowded, disorganized display. Researchers have observed that much user activity is characterized by movement from one set of functionally related windows to another. Bannon *et al.* [54] define the notion of a *workspace* — the grouping together of sets of windows known to be functionally related to some activity or goal — arguing that this kind of organization more closely matches users' goal structure than individual windows [96]. Card *et al.* [140] also found that window usage could be categorized according to a 'working set' model. They looked at the relationship between the demands of the task and the number of windows in use, and found the largest number of individual windows were in use when users transitioned from one task to another.

Based on these and other observations, Henderson and Card [420] built a system intended to make it easier for users to move between 'multiple virtual workspaces' [96]. The system uses a 3D spatial metaphor, where each workspace is a 'room,' and users transition between workspaces by 'moving' through virtual doors. By 'traveling' from one room to the next, users can change from one work context to another. In each work context, the application programs and data files that are associated with that work context are visible and readily available for reopening and perusal. The workspace notion as developed by Card *et al.* also emphasizes the importance of having sessions persist across time. The user should be able to leave a room dedicated to some task, work on another task, and three days later return to the first room and see all of the applications still in the same state as before. This notion of bundling applications and data together for each task has since been widely adopted by window manager software in workstation operating system interfaces.

Elastic windows [428] is an extension to the workspace or rooms notion to the organization of 2D tiled windows. The main idea is to make the transition easier from one role or task to another, by adjusting how much of the screen real estate is consumed by the current role. The user can enlarge an entire group of windows with a simple gesture, and this resizing automatically causes the rest of the workspaces to reduce in size so they all still fit on the screen without overlap.

10.8.3 Example Systems

The following sections describe the information layout and management approaches taken by several modern information access interfaces.

The InfoGrid Layout

The InfoGrid system [667] is a typical example of a monolithic layout for an information access interface. The layout assumes a large display is available

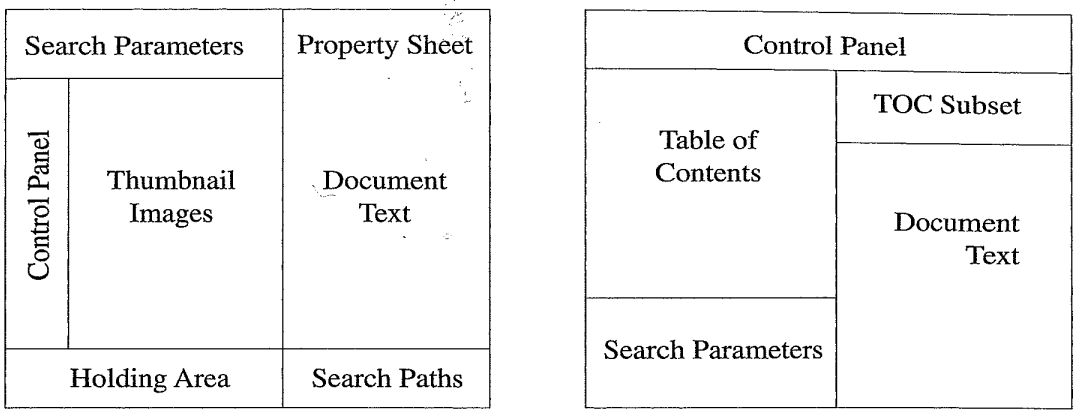


Figure 10.28 Diagrams of monolithic layouts for information access interfaces.

and is divided into a left-hand and right-hand side (see Figure 10.28). The left-hand side is further subdivided into an area at the top that contains structured entry forms for specifying the properties of a query, a column of iconic controls lining the left side, and an area for retaining documents of interest along the bottom. The main central area is used for the viewing of retrieval results, either as thumbnail representations of the original documents, or derived organizations of the documents, such as Scatter/Gather-style cluster results. Users can select documents from this area and store them in the holding area below or view them in the right-hand side. Most of the right-hand side of the display is used for viewing selected documents, with the upper portion showing metadata associated with the selected document. The area below the document display is intended to show a graphical history of earlier interactions.

Designers must make decisions about which kinds of information to show in the primary view(s). If InfoGrid were used on a smaller display, either the document viewing area or the retrieval results viewing area would probably have to be shown via a pop-up overlapping window; otherwise the user would have to toggle between the two views. If the system were to suggest terms for relevance feedback, one of the existing views would have to be supplanted with this information or a pop-up window would have to be used to display the candidate terms. The system does not provide detailed information for source selection, although this could be achieved in a very simple way with a pop-up menu of choices from the control panel.

The SuperBook Layout

The layout of the InfoGrid is quite similar to that of SuperBook (see section 10.6). The main difference is that SuperBook retains the table of contents-like display in the main left-hand pane, along with indicators of how many documents containing search hits occur in each level of the outline. Like InfoGrid, the main pane of the right-hand side is used to display selected documents. Query

formulation is done just below the table of contents view (although in earlier versions this appeared in a separate window). Terms related to the user's query are shown in this window as well. Large images appear in pop-up overlapping windows.

The SuperBook layout is the result of several cycles of iterative design [481]. Earlier versions used overlapping windows instead of a monolithic layout, allowing users to sweep out a rectangular area on the screen in order to create a new text box. This new text box had its own set of buttons that allowed users to jump to occurrences of highlighted words in other documents or to the table of contents. SuperBook was redesigned after noting results of experimental studies [350, 532] showing that users can be more efficient if given fewer, well chosen interaction paths, rather than allowing wide latitude (A recent study of auditory interfaces found that although users were more efficient with a more flexible interface, they nevertheless preferred the more rigid, predictable interface [801]). The designers also took careful note of log files of user interactions. Before the redesign, users had to choose to view the overall frequency of a hit, move the mouse to the table of contents window, click the button and wait for the results to be updated. Since this pattern was observed to occur quite frequently, in the next version of the interface, the system was redesigned to automatically perform this sequence of actions immediately after a search was run.

The SuperBook designers also attempted a redesign to allow the interface to fit into smaller displays. The redesign made use of small, overlapping windows. Some of the interaction sequences that were found useful in this more constrained environment were integrated into later designs for large monolithic displays.

The DLITE Interface

The DLITE system [193, 192] makes a number of interesting design choices. It splits functionality into two parts: control of the search process and display of results . The control portion is a graphical direct manipulation display with animation (see Figure10.29). Queries, sources, documents, and groups of retrieved documents are represented as graphical objects. The user creates a query by filling out the editable fields within a query constructor object. The system manufactures a query object, which is represented by a small icon which can be dragged and dropped onto iconic representations of collections or search services. If a service is active, it responds by creating an empty results set object and attaching the query to this. A set of retrieval results is represented as a circular pool, and documents within the result set are represented as icons distributed along the perimeter of the pool. Documents can be dragged out of the results set pool and dropped into other services, such as a document summarizer or a language translator. Meanwhile, the user can make a copy of the query icon and drop it onto another search service. Placing the mouse over the iconic representation of the query causes a 'tool-tips' window to pop up to show the contents of the underlying query. Queries can be stored and reused at a later time, thus facilitating retention of previously successful search strategies.

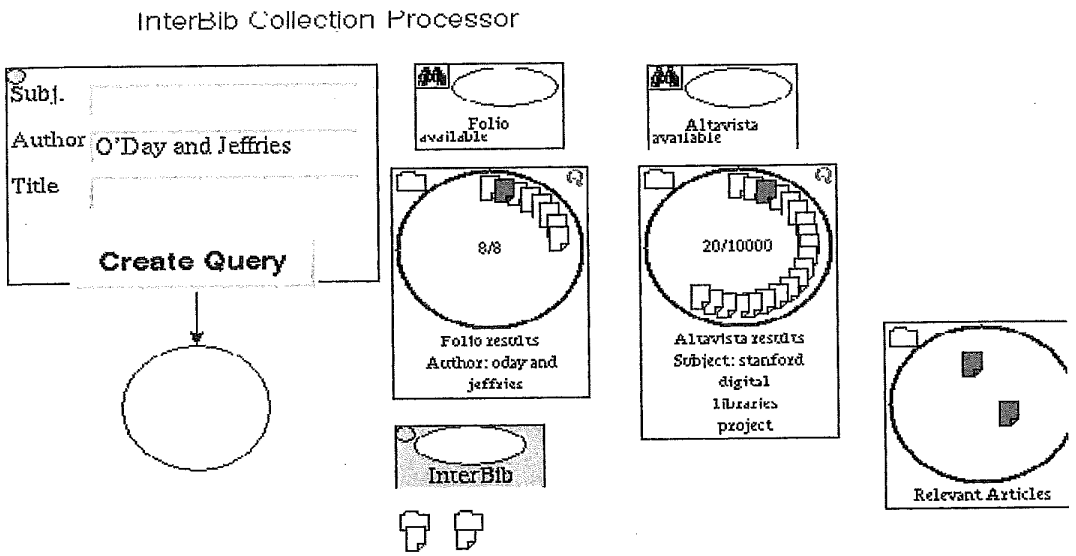


Figure 10.29 The DLITE interface [193].

A flexible interface architecture frees the user from the restriction of a rigid order of commands. On the other hand, as seen in the SuperBook discussion, such an architecture must provide guidelines, to help get the user started, give hints about valid ways to proceed, and prevent the user from making errors. The graphical portion of the DLITE interface makes liberal use of animation to help guide the user. For example, if the user attempts to drop a query in the document summarizer icon — an illegal operation — rather than failing and giving the user an accusatory error message [185], the system takes control of the object being dropped, refusing to let it be placed on the representation for the target application, and moves the object left, right, and left again, mimicking a 'shake-the-head-no' gesture. Animation is also used to help the user understand the state of the system, for example, in showing the progress of the retrieval of search results by moving the result set object away from the service from which it was invoked.

DLITE uses a separate Web browser window for the display of detailed information about the retrieved documents, such as their bibliographic citations and their full text. The browser window is also used to show Scatter/Gather-style cluster results and to allow users to select documents for relevance feedback. Earlier designs of the system attempted to incorporate text display into the direct manipulation portion, but this was found to be infeasible because of the space required [192]. Thus, DLITE separates the control portion of the information access process from the scanning and reading portion. This separation allows for reusable query construction and service selection, while at the same time allowing for a legible view of documents and relationships among retrieved documents. The selection in the display view is linked to the graphical control portion, so a document viewed in the display could be used as part of a query in a query constructor.

DLITE also incorporates the notion of a workspace, or 'workcenter,' as it is known in this system. Different workspaces are created for different kinds of tasks. For example, a workspace for buying computer software can be equipped with source icons representing good sources of reviews of computer software, good Web sites to search for price information and link to the user's online credit service.

The SketchTrieve Interface

The guiding principle behind the SketchTrieve interface [365] is the depiction of information access as an informal process, in which half-finished ideas and partly explored paths can be retained for later use, saved and brought back to compare to later interactions, and the results can be combined via operations on graphical objects and connectors between them. It has been observed [584, 722] that users use the physical layout of information within a spreadsheet to organize information. This idea motivates the design of SketchTrieve, which allows users to arrange retrieval results in a side-by-side manner to facilitate comparison and recombination (see Figure 10.30).

The notion of a canvas or workspace for the retention of the previous context should be adopted more widely in future. Many issues are not easily solved, such as how to show the results of a set of interrelated queries, with minor modifications based on query expansion, relevance feedback, and other forms of modification. One idea is to show sets of related retrieval results as a stack of

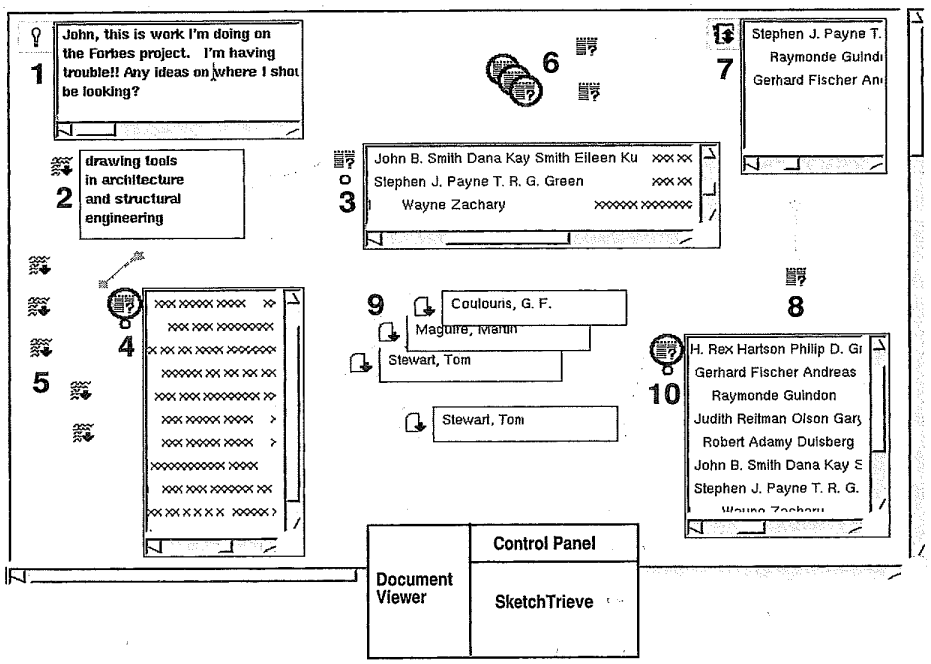


Figure 10.30 The SketchTrieve interface [365].

cards within a folder and allow the user to extract subsets of the cards and view them side by side, as is done in SketchTrieve, or compare them via a difference operation.

10.8.4 Examples of Poor Use of Overlapping Windows

Sometimes conversion from a command-line-based interface to a graphical display can cause problems. Hancock-Beaulieu *et al.* [337] describe poor design decisions made in an overlapping windows display for a bibliographic system. (An improvement was found with a later redesign of the system that used a monolithic interface [336].) Problems can also occur when designers make a 'literal' transformation from a TTY interface to a graphical interface. The consequences can be seen in the current LEXIS-NEXIS interface, which does not make use of the fact that window systems allow the user to view different kinds of information simultaneously. Instead, despite the fact that it occupies the entire screen, the interface does not retain window context when the user switches from one function to another. For example, viewing a small amount of metadata about a list of retrieved titles causes the list of results to disappear, rather than overlaying the information with a pop-up window or rearranging the available space with resizable tiles. Furthermore, this metadata is rendered in poorly-formatted ASCII instead of using the bit-map capabilities of a graphical interface. When a user opts to see the full text view of a document, it is shown in a small space, a few paragraphs at a time, instead of expanding to fill the entire available space.

10.8.5 Retaining Search History

Section 10.3 discusses information seeking strategies and behaviors that have been observed by researchers in the field. This discussion suggests that the user interface should show what the available choices are at any given point, as well as what moves have been made in the past, short-term tactics as well as longer-term strategies, and allow the user to annotate the choices made and information found along the way. Users should be able to bundle search sessions as well as save individual portions of a given search session, and flexibly access and modify each. There is also increasing interest in incorporating personal preference and usage information both into formulation of queries and use of the results of search [277].

For the most part these strategies are not supported well in current user interfaces; however some mechanisms have been introduced that begin to address these needs. In particular, mechanisms to retain prior history of the search are useful for these tasks. Some kind of history mechanism has been made available in most search systems in the past. Usually these consist of a list of the commands executed earlier. More recently, graphical history has been introduced, that allows tracking of commands and results as well. Kim and Hirtle

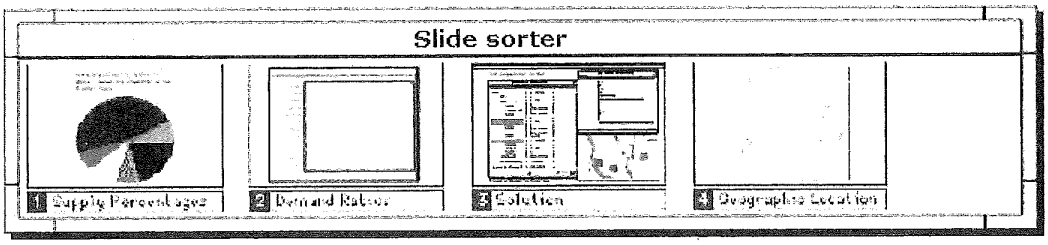


Figure 10.31 The VISAGE interaction history visualization [685].

[440] present a summary of graphical history presentation mechanisms. Recently, a graphical interface that displays Web page access history in a hierarchical structure was found to require fewer page accesses and require less time when returning to pages already visited [370].

An innovation of particular interest for information access interfaces is exemplified by the saving of state in miniature form in a 'slide sorter' view as exercised by the VISAGE system for information visualization [685] (see Figure 10.31). The VISAGE application has the added advantage of being visual in nature and so individual states are easier to recognize. Although intended to be used as a presentation creation facility, this interface should also be useful for retaining search action history.

10.8.6 Integrating Scanning, Selection, and Querying

User interfaces for information access in general do not do a good job of supporting strategies, or even of sequences of movements from one operation to the next. Even something as simple as taking the output of retrieval results from one query and using them as input to another query executed in a later search session is not well supported in most interfaces.

Hertzum and Frokjaer [368] found that users preferred an integration of scanning and query specification in their user interfaces. They did not, however, observe better results with such interactions. They hypothesized that if interactions are too unrestricted this can lead to erroneous or wasteful behavior, and interaction between two different modes requires more guidance. This suggests that more flexibility is needed, but within constraints (this argument was also made in the discussion of the SuperBook system in section 10.6).

There are exceptions. The new Web version of the Melyvl system provides ways to take the output of one query and modify it later for re-execution (see Figure 10.32). The workspace-based systems such as DLITE and Rooms allow storage and reuse of previous state. However, these systems do not integrate the general search process well with scanning and selection of information from auxiliary structures. Scanning, selection, and querying needs to be better integrated in general. This discussion will conclude with an example of an interface that does attempt to tightly couple querying and browsing.

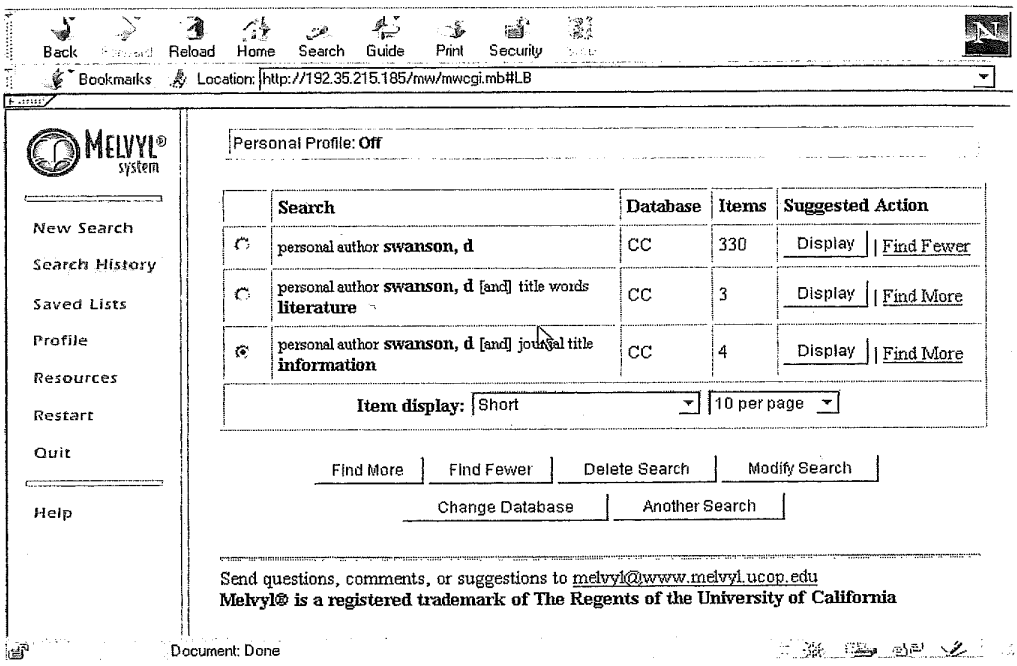


Figure 10.32 A view of query history revision in the Web-based version of the Melvyl bibliographic catalog. Copyright ©, The Regents of the University of California.

The Cat-a-Cone interface integrates querying and browsing of very large category hierarchies with their associated text collections. The prototype system uses 3D+animation interface components from the Information Visualizer [144], applied in a novel way, to support browsing and search of text collections and their category hierarchies. See Figure 10.33. A key component of the interface is the separation of the graphical representation of the category hierarchy from the graphical representation of the documents. This separation allows for a fluid, flexible interaction between browsing and search, and between categories and documents. It also provides a mechanism by which a *set* of categories associated with a document can be viewed along with their hierarchical context.

Another key component of the design is assignment of first-class status to the representation of text content. The retrieved documents are stored in a 3D+animation book representation [144] that allows for compact display of moderate numbers of documents. Associated with each retrieved document is a page of links to the category hierarchy and a page of text showing the document contents. The user can 'ruffle' the pages of the book of retrieval results and see corresponding changes in the category hierarchy, which is also represented in 3D+animation. All and only those parts of the category space that reflect the semantics of the retrieved document are shown with the document.

The system allows for several different kinds of starting points. Users can start by typing in a name of a category and seeing which parts of the category hierarchy match it. For example, Figure 10.34 shows the results of searching on

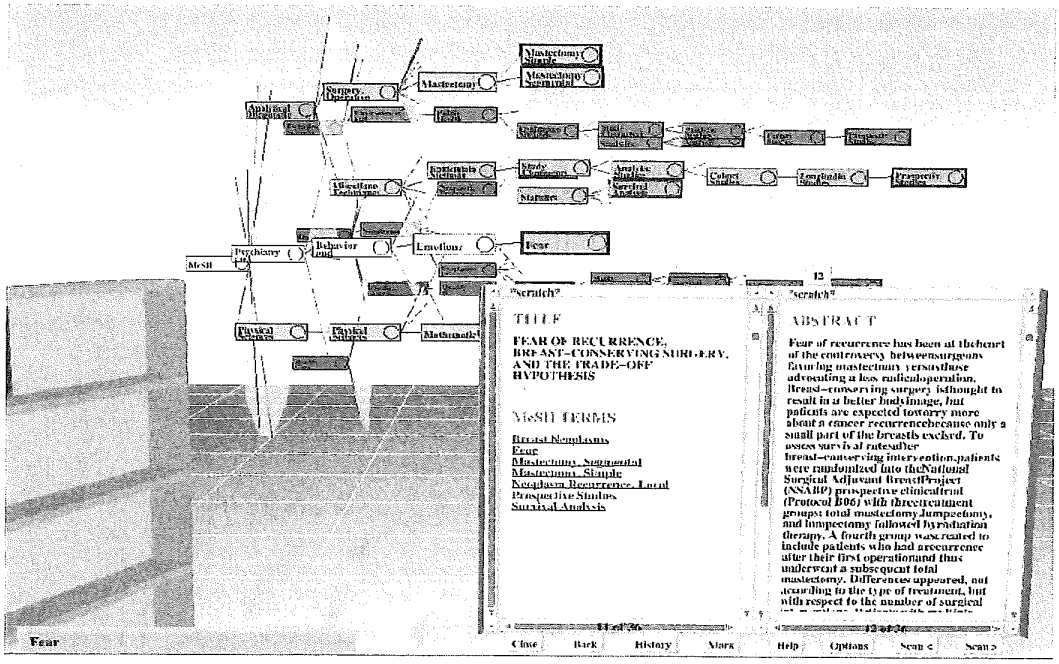


Figure 10.33 The Cat-a-Cone interface for integrating category and text scanning and search [358].

'Radiation' over the MeSH terms in this subcollection. The word appears under four main headings (*Physical Sciences*, *Diseases*, *Diagnostics*, and *Biological Sciences*). The hierarchy immediately shows why 'Radiation' appears under *Diseases* — as part of a subtree on occupational hazards. Now the user can select one or more of these category labels as input to a query specification.

Another way the user can start is by simply typing in a free text query into an entry label. This query is matched against the collection. Relevant documents are retrieved and placed in the book format. When the user 'opens' the book to a retrieved document, the parts of the category hierarchy that correspond to the retrieved documents are shown in the hierarchical representation. Thus, multiple intersecting categories can be shown simultaneously, in their hierarchical context. Thus, this interface fluidly combines large, complex metadata, starting points, scanning, and querying into one interface. The interface allows for a kind of relevance feedback, by suggesting additional categories that are related to the documents that have been retrieved. This interaction model is similar to that proposed by [5].

Recall the evaluation of the Kohonen feature map representation discussed in section 10.4. The experimenters found that some users expressed a desire for a visible hierarchical organization, others wanted an ability to zoom in on a subarea to get more detail, and some users disliked having to look through the entire map to find a theme, desiring an alphabetical ordering instead. The subjects liked the ease of being able to jump from one area to another without

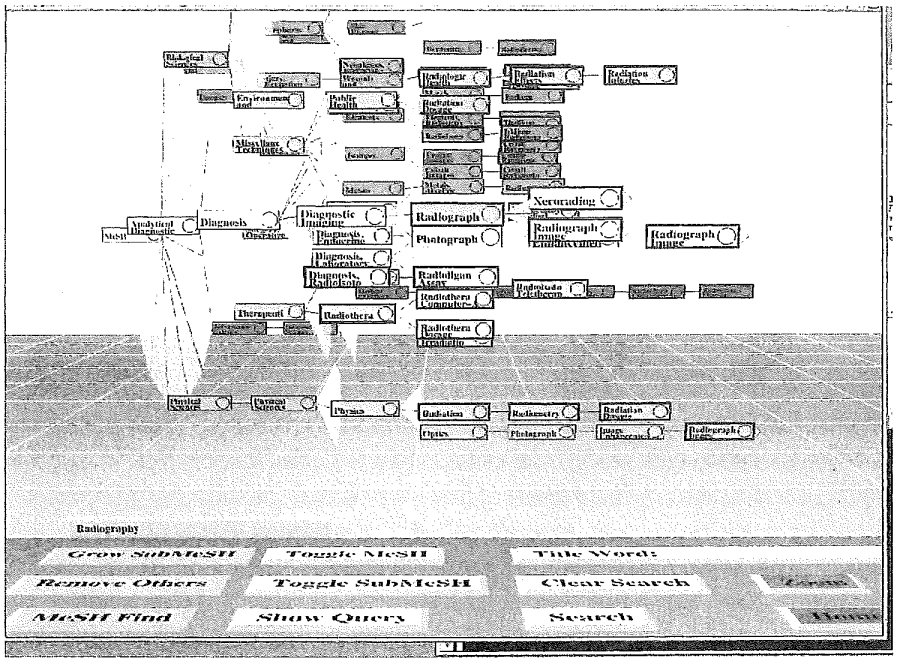


Figure 10.34 An interface for a starting point for searching over category labels [358].

having to back up (as is required in Yahoo!) and liked the fact that the maps have varying levels of granularity.

These results all support the design decisions made in the Cat-a-Cone. Hierarchical representation of term meanings is supported, so users can choose which level of description is meaningful to them. Furthermore, different levels of description can be viewed simultaneously, so more familiar concepts can be viewed in more detail, and less familiar at a more general level. An alphabetical ordering of the categories coupled with a regular expression search mechanism allows for straightforward location of category labels. Retrieved documents are represented as first-class objects, so full text is visible, but in a compact form. Category labels are disambiguated by their ancestor/descendant/sibling representation. Users can jump easily from one category to another and can in addition query on multiple categories simultaneously (something that is not a natural feature of the maps). The Cat-a-Cone has several additional advantages as well, such as allowing a document to be placed at the intersection of several categories, and explicitly linking document contents with the category representation.

10.9 Trends and Research Issues

The importance of human computer interaction is receiving increasing recognition within the field of computer science [587]. As should be evident from the

contents of this chapter, the role of the user interface in the information access process has only recently begun to receive the attention it deserves. Research in this area can be expected to increase rapidly, primarily because of the rise of the Web. The Web has suddenly made vast quantities of information available globally, leading to an increase in interest in the problem of information access. This has lead to the creation of new information access paradigms, such as the innovative use of relevance feedback seen in the Amazon.com interface. Because the Web provides a platform-independent user interface, investment in better user interface design can have an impact on a larger user population than before.

Another trend that can be anticipated is an amplified interest in organization and search over personal information collections. Many researchers are proposing that in future a person's entire life will be recorded using various media, from birth to death. One motivation for this scenario is to enable searching over everything a person has ever read or written. Another motivation is to allow for searching using contextual clues, such as 'find the article I was reading in the meeting I had on May 1st with Pam and Hal'. If this idea is pursued, it will require new, more sophisticated interfaces for searching and organizing a huge collection of personal information.

There is also increasing interest in leveraging the behavior of individuals and groups, both for rating and assessing the quality of information items, and for suggesting starting points for search within information spaces. Recommender systems can be expected to increase in prevalence and diversity. User interfaces will be needed to guide users to appropriate recommended items based on their information needs.

The field of information visualization needs some new ideas about how to display large, abstract information spaces intuitively. Until this happens, the role of visualization in information access will probably be primarily confined to providing thematic overviews of topic collections and displaying large category hierarchies dynamically. Breakthroughs in information visualization can be expected to have a strong impact on information access systems.

10.10 Bibliographic Discussion

The field of human-computer interaction is a broad one, and this chapter touches on only a small subset of pertinent issues. For further information, see the excellent texts on user interface design by Shneiderman [725], information seeking behavior by Marchionini [542], and digital libraries by Lesk [501]. An excellent book on visual design is that of Mullet and Sano [580]. Tufte has written thought-provoking and visually engaging books on the power of information visualization [769, 770] and a collection of papers on information visualization has been edited by Card *et al.* [141].

This chapter has discussed many ideas for improving the human-computer interaction experience for information seekers. This is the most rapidly

developing area of information access today, and improvements in the interface are likely to lead the way toward better search results and better-enabled information creators and users. Research in the area of human-computer interaction is difficult because the field is relatively new, and because it can be difficult to obtain strong results when running user studies. These challenges should simply encourage those who really want to influence the information access systems of tomorrow.

Acknowledgements

The author gratefully acknowledges the generous and helpful comments on the contents of this chapter by Gary Marchionini and Ben Shneiderman, the excellent administrative assistance of Barbara Goto, and the great faith and patience of Ricardo Baeza-Yates and Berthier Ribeiro-Neto.

The user can also refine the query by constructing more complex queries based on the previous answer.

The Web pages retrieved by the search engine in response to a user query are ranked, usually using statistics related to the terms in the query. In some cases this may not have any meaning, because relevance is not fully correlated with statistics about term occurrence within the collection. Some search engines also taking into account terms included in metatags or the title, or the popularity of a Web page to improve the ranking. This topic is covered next.

13.4.4 Ranking

Most search engines use variations of the Boolean or vector model (see Chapter 2) to do ranking. As with searching, ranking has to be performed without accessing the text, just the index. There is not much public information about the specific ranking algorithms used by current search engines. Further, it is difficult to compare fairly different search engines given their differences, and continuous improvements. More important, it is almost impossible to measure recall, as the number of relevant pages can be quite large for simple queries. Some inconclusive studies include [327, 498].

Yuwono and Lee [844] propose three ranking algorithms in addition to the classical tf-idf scheme (see Chapter 2). They are called Boolean spread, vector spread, and most-cited. The first two are the normal ranking algorithms of the Boolean and vector model extended to include pages pointed to by a page in the answer or pages that point to a page in the answer. The third, most-cited, is based only on the terms included in pages having a link to the pages in the answer. A comparison of these techniques considering 56 queries over a collection of 2400 Web pages indicates that the vector model yields a better recall-precision curve, with an average precision of 75%.

Some of the new ranking algorithms also use hyperlink information. This is an important difference between the Web and normal IR databases. The number of hyperlinks that point to a page provides a measure of its popularity and quality. Also, many links in common between pages or pages referenced by the same page often indicates a relationship between those pages. We now present three examples of ranking techniques that exploit these facts, but they differ in that two of them depend on the query and the last does not.

The first is WebQuery [148], which also allows visual browsing of Web pages. WebQuery takes a set of Web pages (for example, the answer to a query) and ranks them based on how connected each Web page is. Additionally, it extends the set by finding Web pages that are highly connected to the original set. A related approach is presented by Li [512].

A better idea is due to Kleinberg [444] and used in HITS (Hypertext Induced Topic Search). This ranking scheme depends on the query and considers the set of pages S that point to or are pointed by pages in the answer. Pages that have many links pointing to them in S are called authorities (that is, they should have relevant content). Pages that have many outgoing links are called hubs (they should point to similar content). A positive two-way feedback exists:

better authority pages come from incoming edges from good hubs and better hub pages come from outgoing edges to good authorities. Let $H(p)$ and $A(p)$ be the hub and authority value of page p. These values are defined such that the following equations are satisfied for all pages p:

$$H(p) = \sum_{u \in S \mid p \to u} A(u) , \qquad A(p) = \sum_{v \in S \mid v \to p} H(v)$$

where $H(p)$ and $A(p)$ for all pages are normalized (in the original paper, the sum of the squares of each measure is set to one). These values can be determined through an iterative algorithm, and they converge to the principal eigenvector of the link matrix of S. In the case of the Web, to avoid an explosion of the size of S, a maximal number of pages pointing to the answer can be defined. This technique does not work with non-existent, repeated, or automatically generated links. One solution is to weight each link based on the surrounding content. A second problem is that the topic of the result can become diffused. For example, a particular query is enlarged by a more general topic that contains the original answer. One solution to this problem is to analyze the content of each page and assign a score to it, as in traditional IR ranking. The link weight and the page score can be included on the previous formula multiplying each term of the summation [154, 93, 153]. Experiments show that the recall and precision on the first ten answers increases significantly [93]. The order of the links can also be used by dividing the links into subgroups and using the HITS algorithm on those subgroups instead of the original Web pages [153].

The last example is PageRank, which is part of the ranking algorithm used by Google [117]. PageRank simulates a user navigating randomly in the Web who jumps to a random page with probability q or follows a random hyperlink (on the current page) with probability $1 - q$. It is further assumed that this user never goes back to a previously visited page following an already traversed hyperlink backwards. This process can be modeled with a Markov chain, from where the stationary probability of being in each page can be computed. This value is then used as part of the ranking mechanism of Google. Let $C(a)$ be the number of outgoing links of page a and suppose that page a is pointed to by pages p_1 to p_n. Then, the PageRank, $PR(a)$ of a is defined as

$$PR(a) = q + (1 - q) \sum_{i=1}^{n} PR(p_i)/C(p_i)$$

where q must be set by the system (a typical value is 0.15). Notice that the ranking (weight) of other pages is normalized by the number of links in the page. PageRank can be computed using an iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the Web (which is the transition matrix of the Markov chain). Crawling the Web using this ordering has been shown to be better than other crawling schemes [168] (see next section).

Information Retrieval/Database Management

Modern Information Retrieval

Baeza-Yates Ribeiro-Neto

We live in the information age, where swift access to relevant information in whatever form or medium can dictate the success or failure of businesses or individuals. The timely provision of relevant information with minimal 'noise' is critical to modern society and this is what information retrieval (IR) is all about. It is a dynamic subject, with current changes driven by the expansion of the World Wide Web, the advent of modern and inexpensive graphical user interfaces and the development of reliable and low-cost mass storage devices. *Modern Information Retrieval* discusses all these changes in great detail and can be used for a first course on IR as well as graduate courses on the topic.

The organization of the book, which includes a comprehensive glossary and is supported by a web page (www.dcc.ufmg.br/irbook or sunsite.dcc.uchile.cl/irbook), allows the reader to either obtain a broad overview or detailed knowledge of all the key topics in modern IR. The heart of the book is the nine chapters written by Baeza-Yates and Ribeiro-Neto, two leading exponents in the field. For those wishing to delve deeper into key areas there are further state-of-the-art chapters on specialized topics, fully integrated and using the same notation and terminology, written by leading researchers:

- **Parallel and Distributed IR** - algorithms and architectures (Eric Brown)
- **User Interfaces and Visualization** - the main interface paradigms for query formation and visualization of results (Marti A. Hearst)
- **Multimedia IR: Models and Languages** - including MULTOS and SQL3 (Elisa Bertino, Barbara Catania and Elena Ferrari)
- **Multimedia IR: Indexing and Searching** - R-trees and GEMINI and QBIC (Christos Faloutsos)
- **Libraries and Bibliographical Systems** - online systems and public access catalogs (Edie M. Rasmussen)
- **Digital Libraries** - the main challenges for effective deployment (Edward A. Fox and Ohm Sornil)

In addition, the book contains several chapters on

- **Text IR** - all the main IR models, query operations, text operations, indexing and searching (three of them co-authored with Gonzalo Navarro or Nivio Ziviani)
- **The Web** - challenges, measures and models, search engines, directories, query languages, metasearches and trends

Modern Information Retrieval will prove invaluable to students in Computer Science, Information Science and Library Science departments, as well as to programmers and analysts working with products related to the Web, intranets, document database systems and digital libraries.

Visit us on the World Wide Web at:
http://www.pearsoneduc.com
http://www.awl-he.com/computing/
http://www.awl.com/cseng/

ISBN 0-201-39829-X

9 780201 398298 >

ADDISON-WESLEY

acm PRESS