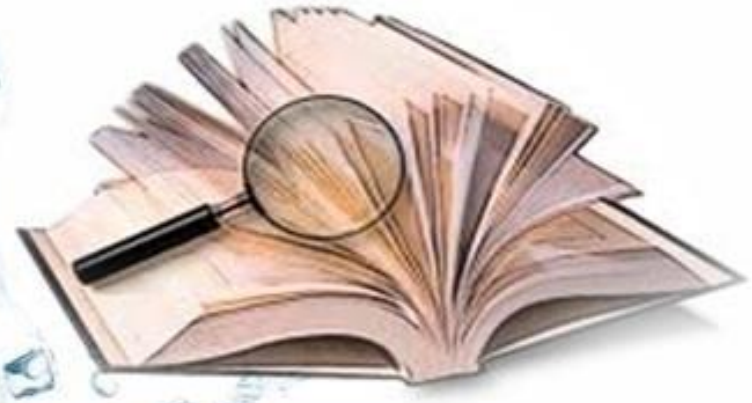
KARNATAKA STATE OPEN UNIVERSITY

MUKTHAGANGOTRI, MYSORE- 570 006

DEPARTMENT OF STUDIES IN INFORMATION TECHNOLOGY

M.Sc IN INFORMATION SCIENCE

II SEMESTER



INFORMATION ORGANIZATION AND RETRIEVAL

IS 2.3 BLOCK 1 TO 4

IS 2.3

INFORMATION ORGANIZATION AND RETRIEVAL

PREFACE

The objectives of information retrieval (IR) systems are to reduce the complexity of handling voluminous data, what has been popularly projected as "information overload". The application of IR systems found in different educational sectors mainly in many universities and public libraries that use IR systems to provide access to books, journals and other documents. The Web search engines are the most visible IR applications used by common man for many of his needs. In this context, the study of IR system development is very useful to the student community for developing efficient IR systems. The students are expected to learn the importance of the Information retrieval and its organization along its working principle with respect to many real world problems. The ultimate aim of IR system is to understand its working principle and to build intelligent software and tools that works better than the performance of humans in retrieving desired information. On their way towards this goal, many commercial IR systems have been developed for quite number of different applications.

We introduce basic concepts and models of Information retrieval system from a computer science perspective. The focus of the course will be on the study of different models of information retrieval systems, data structures used in the design of IR system, search issues, document and term clustering techniques. Different types of information retrieval system are also discussed which find applications in diversified fields. This course will empower the students to know how to design IR systems using classical Boolean model and in depth analysis is provided to design multimedia based IR systems.

This concise text book provides an accessible introduction to information retrieval and organization that supports a foundation or module course on Information organization and its retrieval covering a broad selection of the sub-disciplines within this field. The textbook presents concrete algorithms and applications in the areas of digital library, web search, multimedia databases etc.

Organization of the material: The book introduces its topics in ascending order of complexity and is divided into four modules, containing four units each.

In the first module, we begin with an introduction to information retrieval highlighting its applications and techniques. Different types of IR systems, the architecture of IR system, data structures used in the design of IR system are addressed. The search issues that associate with information retrieval system design are also discussed in brief.

In the second module, we discussed the capabilities of IR system followed by the usage of advanced data structures for efficient retrieval of information from a large repository of data is presented. The indexing concepts to speed up the retrieval process are presented in detail.

The third module contains descriptions on different types of information retrieval system where much discussion is on multimedia database systems and their architecture with performance issues. The concept of term clustering and document clustering used in the design of web applications is presented followed by discussion on text search algorithms used in the design of web search engines.

In the fourth module, the concepts of relevance feedback which is an alternative to thesaurus expansion to assist the user in creating a search statement are presented. In addition, the information visualization concepts are discussed followed by the issues related to the evaluation of Information Retrieval Systems which is essential to understand the source of weaknesses in existing systems and tradeoffs between using different algorithms are presented in detail.

Every unit covers a distinct problem and includes a quick summary at the end, which can be used as a reference material while reading information retrieval and organization. Much of the material found here is interesting as a view into how the IR works, even if you do not need it for a specific works.

Happy reading to all the students!!!

Karnataka State Open University

Manasagangothri, Mysore – 570 006

Second Semester M.Sc in Information Science

Information Organization and Retrieval

Module 1

Module 2

Module 3

Module 4

Course Design and Editorial Committee

Prof. M.|G.Krishnan Vice Chancellor & Chairperson Karnataka State Open University Manasagangotri, Mysore – 570 006	**Prof. Vikram Raj Urs** Dean (Academic) & Convener Karnataka State Open University Manasagangotri, Mysore – 570 006
Head of the Department	**Course Co-Ordinator**
Rashmi B.S Assistant professor & Chairperson DoS in Information Technology Karnataka State Open University Manasagangotri, Mysore – 570 006	**Mr. Mahesha DM** Assistant professor in Computer Science DoS in Computer Science Karnataka State Open University Manasagangotri, Mysore – 570 006

Course Editor

Ms. Nandini H.M
Assistant professor of Information Technology
DoS in Information Technology
Karnataka State Open University
Manasagangotri, Mysore – 570 006

Course Writers

Dr. B. H. Shekar Associate Professor & Chairman Department of Computer Science Manasagangotri Mangalore University	Dr. Manjaiah D H Associate Professor Department of Computer Science Manasagangotri Mangalore University

Publisher

Registrar
Karnataka State Open University
Manasagangotri, Mysore – 570 006

Developed by Academic Section, KSOU, Mysore
Karnataka State Open University, 2012

Printed and Published on behalf of Karnataka State Open University, Mysore-6 by the **Registrar (Administration)**

UNIT 1: BOOLEAN MODEL BASED INFORMATION RETRIEVAL SYSTEM

Structure

1.0 INTRODUCTION

Information retrieval (IR) is a broad area of Computer Science focused primarily on providing the users with easy access to information of their interest, as follows.

> *Information retrieval deals with the representation, storage, organization of, and access to information items such as documents, Web pages, online catalogues, structured and semi-structured records, multimedia objects. The representation and organization of the information items should be such as to provide the users with easy access to information of their interest.*

Information retrieval is the area of study concerned with searching for documents, for information within documents, and for metadata about documents, as well as that of searching structured storage, relational databases, and the World Wide Web. There is overlap in the usage of the terms data retrieval, document retrieval, information retrieval, and text retrieval, but each also has its own body of literature, theory, praxis, and technologies. IR is interdisciplinary, based on computer science, mathematics, library science, information science, information architecture, cognitive psychology, linguistics, statistics and law.

Automated information retrieval systems are used to reduce what has been called "information overload". Many universities and public libraries use IR systems to provide access to books, journals and other documents. Web search engines are the most visible IR applications.

An information retrieval process begins when a user enters a query into the system. Queries are formal statements of information needs, for example search strings in web search engines. In information retrieval a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy.

An object is an entity that is represented by information in a database. User queries are matched against the database information. Depending on the application the data objects may be, for example, text documents, images, audio, mind maps or videos. Often the documents themselves are not kept or stored directly in the IR system, but are instead represented in the system by document surrogates or metadata.

Most IR systems compute a numeric score on how well each object in the database matches the query, and rank the objects according to this value. The top ranking objects are then shown to the user. The process may then be iterated if the user wishes to refine the query.

1.1 HISTORY OF INFORMATION SYSTEMS

The idea of using computers to search for relevant pieces of information was popularized in the article ***As We May Think*** by Vannevar Bush in 1945. The first automated information retrieval systems were introduced in the 1950s and 1960s. By 1970 several different techniques had been shown to perform well on small text corpora such as the Cranfield collection (several thousand documents). Large-scale retrieval systems, such as the Lockheed Dialog system, came into use early in the 1970s.

In 1992, the US Department of Defence along with the National Institute of Standards and Technology (NIST) co-sponsored the Text Retrieval Conference (TREC) as part of the TIPSTER text program. The aim of this was to look into the information retrieval community by supplying the infrastructure that was needed for evaluation of text retrieval methodologies on a very large text collection. This catalyzed research on methods that scale to huge corpora. The introduction of web search engines has boosted the need for very large scale retrieval systems even further.

The use of digital methods for storing and retrieving information has led to the phenomenon of digital obsolescence, where a digital resource ceases to be readable because the physical media, the reader required to read the media, the hardware, or the software that runs on it, is no longer available. The information is initially easier to retrieve than if it were on paper, but is then effectively lost.

The timeline of Information system development is given below.

Before the 1900s

- 1801: Joseph Marie Jacquard invents the Jacquard loom, the first machine to use punched cards to control a sequence of operations.
- 1880s: Herman Hollerith invents an electro-mechanical data tabulator using punch cards as a machine readable medium.
- 1890 Hollerith cards, keypunches and tabulators used to process the 1890 US Census data.

1920s-1930s

- Emanuel Goldberg submits patents for his "Statistical Machine" a document search engine that used photoelectric cells and pattern recognition to search the metadata on rolls of microfilmed documents.

1940s–1950s

- Late 1940s: The US military confronted problems of indexing and retrieval of wartime scientific research documents captured from Germans.
- 1945: Vannevar Bush's As We May Think appeared in Atlantic Monthly.
- 1947: Hans Peter Luhn (research engineer at IBM since 1941) began work on a mechanized punch card-based system for searching chemical compounds.
- 1950s: Growing concern in the US for a "science gap" with the USSR motivated, encouraged funding and provided a backdrop for mechanized literature searching systems (Allen Kent et al.) and the invention of citation indexing (Eugene Garfield).
- 1950: The term "information retrieval" appears to have been coined by Calvin Mooers[2].
- 1951: Philip Bagley conducted the earliest experiment in computerized document retrieval in a master thesis at MIT.[3]
- 1955: Allen Kent joined Case Western Reserve University, and eventually became associate director of the Center for Documentation and Communications Research. That same year, Kent and colleagues published a paper in American Documentation describing the precision and recall measures as well as detailing a proposed "framework" for evaluating an IR system which included statistical sampling methods for determining the number of relevant documents not retrieved.

- 1958: International Conference on Scientific Information Washington DC included consideration of IR systems as a solution to problems identified. See: Proceedings of the International Conference on Scientific Information, 1958 (National Academy of Sciences, Washington, DC, 1959)
- 1959: Hans Peter Luhn published "Auto-encoding of documents for information retrieval."

1960s:

- Early 1960s: Gerard Salton began work on IR at Harvard, later moved to Cornell.
- 1960: Melvin Earl (Bill) Maron and John Lary Kuhns[4] published "On relevance, probabilistic indexing, and information retrieval" in the Journal of the ACM 7(3):216–244, July 1960.
- 1962: Cyril W. Cleverdon published early findings of the Cranfield studies, developing a model for IR system evaluation. See: Cyril W. Cleverdon, "Report on the Testing and Analysis of an Investigation into the Comparative Efficiency of Indexing Systems". Cranfield Collection of Aeronautics, Cranfield, England, 1962.
    - Kent published Information Analysis and Retrieval.
- 1963: Weinberg report "Science, Government and Information" gave a full articulation of the idea of a "crisis of scientific information." The report was named after Dr. Alvin Weinberg.
- Joseph Becker and Robert M. Hayes published text on information retrieval. Becker, Joseph; Hayes, Robert Mayo. Information storage and retrieval: tools, elements, theories. New York, Wiley (1963).
- 1964: Karen Spärck Jones finished her thesis at Cambridge, Synonymy and Semantic Classification, and continued work on computational linguistics as it applies to IR.
- The National Bureau of Standards sponsored a symposium titled "Statistical Association Methods for Mechanized Documentation." Several highly significant papers, including G. Salton's first published reference (we believe) to the SMART system.
- mid-1960s: National Library of Medicine developed MEDLARS Medical Literature Analysis and Retrieval System, the first major machine-readable database and batch-retrieval system.
- Project Intrex at MIT.
- 1965: J. C. R. Licklider published Libraries of the Future.
- 1966: Don Swanson was involved in studies at University of Chicago on Requirements for Future Catalogs.
- late 1960s: F. Wilfrid Lancaster completed evaluation studies of the MEDLARS system and published the first edition of his text on information retrieval.

- 1968:
- Gerard Salton published Automatic Information Organization and Retrieval.
- John W. Sammon, Jr.'s RADC Tech report "Some Mathematics of Information Storage and Retrieval..." outlined the vector model.
- 1969: Sammon's "A nonlinear mapping for data structure analysis" (IEEE Transactions on Computers) was the first proposal for visualization interface to an IR system.

1970s

- early 1970s:
  - First online systems—NLM's AIM-TWX, MEDLINE; Lockheed's Dialog; SDC's ORBIT.
  - Theodor Nelson promoting concept of hypertext, published Computer Lib/Dream Machines.
- 1971: Nicholas Jardine and Cornelis J. van Rijsbergen published "The use of hierarchic clustering in information retrieval", which articulated the "cluster hypothesis." (Information Storage and Retrieval, 7(5), pp. 217–240, December 1971)
- 1975: Three highly influential publications by Salton fully articulated his vector processing framework and term discrimination model:
  - A Theory of Indexing (Society for Industrial and Applied Mathematics)
  - A Theory of Term Importance in Automatic Text Analysis (JASIS v. 26)
  - A Vector Space Model for Automatic Indexing (CACM 18:11)
- 1978: The First ACM SIGIR conference.
- 1979: C. J. van Rijsbergen published Information Retrieval (Butterworths). Heavy emphasis on probabilistic models.

1980s

- 1980: First international ACM SIGIR conference, joint with British Computer Society IR group in Cambridge.
- 1982: Nicholas J. Belkin, Robert N. Oddy, and Helen M. Brooks proposed the ASK (Anomalous State of Knowledge) viewpoint for information retrieval. This was an important concept, though their automated analysis tool proved ultimately disappointing.
- 1983: Salton (and Michael J. McGill) published Introduction to Modern Information Retrieval (McGraw-Hill), with heavy emphasis on vector models.
- 1985: Blair and Maron publish: An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System
- mid-1980s: Efforts to develop end-user versions of commercial IR systems.
  - 1985–1993: Key papers on and experimental systems for visualization interfaces.

- Work by Donald B. Crouch, Robert R. Korfhage, Matthew Chalmers, Anselm Spoerri and others.

- 1989: First World Wide Web proposals by Tim Berners-Lee at CERN.

1990s

- 1992: First TREC conference.
- 1997: Publication of Korfhage's Information Storage and Retrieval[5] with emphasis on visualization and multi-reference point systems.
- Late 1990s: Web search engines implementation of many features formerly found only in experimental IR systems. Search engines become the most common and maybe best instantiation of IR models, research, and implementation.

1.2 INFORMATION SYSTEM MODELS

For the information retrieval to be efficient, the documents are typically transformed into a suitable representation. There are several representations. In figure 1.1,we illustrate the relationship of some common models. In the figure, the models are categorized according to two dimensions: the mathematical basis and the properties of the model.

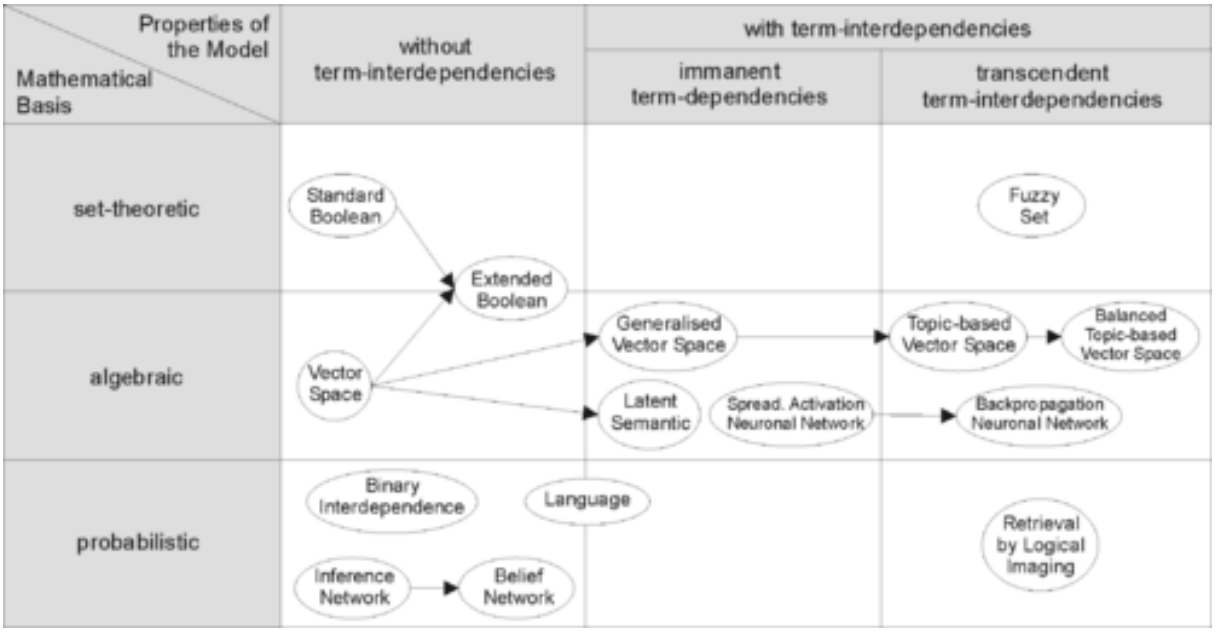


Fig. 1.1. Information Retrieval Categorization (Courtesy: Dominik Kuropka)

First dimension: Mathematical basis

Set-theoretic models represent documents as sets of words or phrases. Similarities are usually derived from set-theoretic operations on those sets. Common models are:

- Standard Boolean model
- Extended Boolean model
- Fuzzy retrieval

Algebraic models represent documents and queries usually as vectors, matrices, or tuples. The similarity of the query vector and document vector is represented as a scalar value.

- Vector space model
- Generalized vector space model
- (Enhanced) Topic-based Vector Space Model
- Extended Boolean model
- Latent semantic indexing aka latent semantic analysis

Probabilistic models treat the process of document retrieval as a probabilistic inference. Similarities are computed as probabilities that a document is relevant for a given query. Probabilistic theorems like the Bayes' theorem are often used in these models.

- Binary Independence Model
- Probabilistic relevance model on which is based the okapi (BM25) relevance function
- Uncertain inference
- Language models
- Divergence-from-randomness model
- Latent Dirichlet allocation

Feature-based retrieval models view documents as vectors of values of feature functions (or just features) and seek the best way to combine these features into a single relevance score, typically by learning to rank methods. Feature functions are arbitrary functions of document and query, and as such can easily incorporate almost any other retrieval model as just a yet another feature.

Second dimension: properties of the model

Models without term-interdependencies treat different terms/words as independent. This fact is usually represented in vector space models by the orthogonality assumption of term vectors or in probabilistic models by an independency assumption for term variables.
Models with immanent term interdependencies allow a representation of interdependencies between terms. However the degree of the interdependency between two terms is defined by

the model itself. It is usually directly or indirectly derived (e.g. by dimensional reduction) from the co-occurrence of those terms in the whole set of documents.

Models with transcendent term interdependencies allow a representation of interdependencies between terms, but they do not allege how the interdependency between two terms is defined. They relay an external source for the degree of interdependency between two terms. (For example a human or sophisticated algorithms.)

1.3 INFORMATION RETRIEVAL USING BOOLEAN MODELS

Mathematical models are used in many scientific areas with the objective to understand and reason about some behaviour or phenomenon in the real world. One might for instance think of a model of our solar system that predicts the position of the planets on a particular date, or one might think of a model of the world climate that predicts the temperature given the atmospheric emissions of greenhouse gases. A model of information retrieval predicts and explains what a user will and relevant given the user query. Models can serve as a blueprint to implement an actual retrieval system.

The Boolean model is the first model of information retrieval and probably also the most criticised model. The model can be explained by thinking of a query term as an unambiguous definition of a set of documents. For instance, the query term economic simply defines the set of all documents that are indexed with the term economic. Using the operators of George Boole's mathematical logic, query terms and their corresponding sets of documents can be combined to form new sets of documents. Boole defined three basic operators, the logical product called AND, the logical sum called OR and the logical difference called NOT. Combining terms with the AND operator will define a document set that is smaller than or equal to the document sets of any of the single terms. For instance, the query social AND economic will produce the set of documents that are indexed both with the term social and the term economic, i.e. the intersection of both sets. Combining terms with the OR operator will define a document set that is bigger than or equal to the document sets of any of the single terms. So, the query social OR political will produce the set of documents that are indexed with either the term social or the term political, or both, i.e. the union of both sets. This is visualised in the Venn diagrams of Figure 1.1 in which each set of documents is visualised by a disc. The intersections of these discs and their complements divide the document collection into 8 non-overlapping regions, the unions of which give 256 different Boolean combinations of `social, political and economic documents'. In Figure 1.1, the

retrieved sets are visualised by the shaded areas. An advantage of the Boolean model is that it gives (expert) users a sense of control over the system.

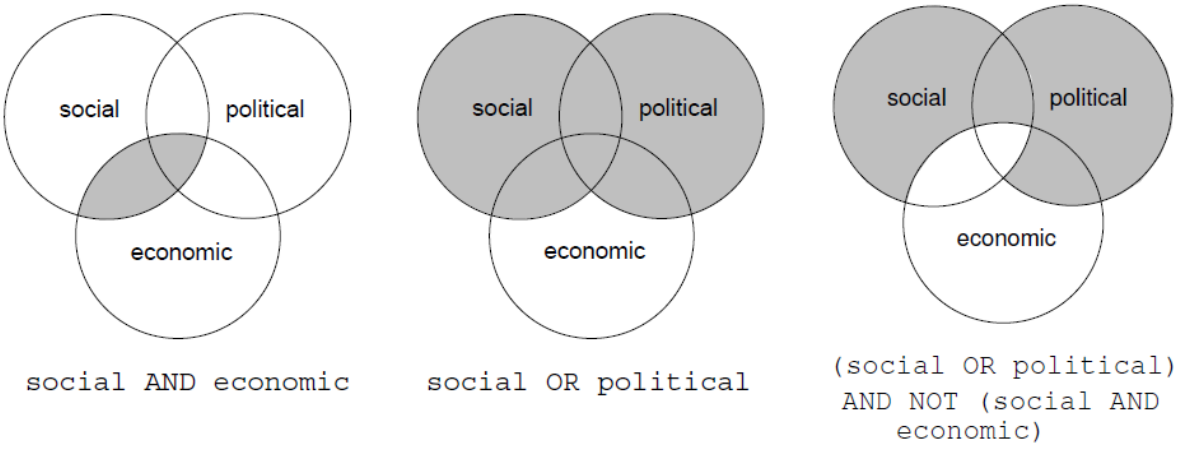


Figure 1.1: Boolean combinations of sets visualised as Venn diagrams.

It is immediately clear why a document has been retrieved given a query. If the resulting document set is either too small or too big, it is directly clear which operators will produce respectively a bigger or smaller set. For untrained users, the model has a number of clear disadvantages. Its main disadvantage is that it does not provide a ranking of retrieved documents. The model either retrieves a document or not, which might lead to the system make rather frustrating decisions. For instance, the query social AND worker AND union will of course not retrieve a document indexed with party, birthday and cake, but will likewise not retrieve a document indexed with social and worker that lacks the term union. Clearly, it is likely that the latter document is more useful than the former, but the model has no means to make the distinction.

Example:

Let the set of original (real) documents be, for example O = {O1, O2, O3}

Where

- O1 = Bayes' Principle: The principle that, in estimating a parameter, one should initially assume that each possible value has equal probability (a uniform prior distribution).
- O2 = Bayesian Decision Theory: A mathematical theory of decision-making which presumes utility and probability functions, and according to which the act to be chosen is the Bayes act, i.e. the one with highest Subjective Expected Utility. If one

had unlimited time and calculating power with which to make every decision, this procedure would be the best way to make any decision.

- O3 = Bayesian Epistemology: A philosophical theory which holds that the epistemic status of a proposition (i.e. how well proven or well established it is) is best measured by a probability and that the proper way to revise this probability is given by Bayesian conditionalisation or similar procedures. A Bayesian epistemologist would use probability to define, and explore the relationship between, concepts such as epistemic status, support or explanatory power.

Let the set T of terms be: T = {t1 = Bayes' Principle, t2 = probability, t3 = decision-making, t4 = Bayesian Epistemology}

Then, the set D of documents is as follows: D = {D1, D2, D3}

where

- D1 = {Bayes' Principle, probability}
- D2 = {probability, decision-making}
- D3 = {probability, Bayesian Epistemology}

Let the query Q be: ***Q = probability AND decision-making***

1. Firstly, the following sets S1 and S2 of documents Di are obtained (retrieved):

- S1 = {D1, D2, D3}
- S2 = {D2}

2. Finally, the following documents Di are retrieved in response to Q: {D1, D2, D3} INTERSECTION {D2} = {D2}

This means that the original document O2 (corresponding to D2) is the answer to Q. Obviously, if there is more than one document with the same representation, every such document is retrieved. Such documents are, in the BIR, indistinguishable (or, in other words, equivalent).

The standard Boolean approach has the following strengths:

- It is easy to implement and it is computationally efficient. Hence, it is the standard model for the current large-scale, operational retrieval systems and many of the major on-line information services use it.
- It enables users to express structural and conceptual constraints to describe important linguistic features. Users find that synonym specifications (reflected by OR-clauses) and phrases (represented by proximity relations) are useful in the formulation of queries.

- The Boolean approach possesses a great expressive power and clarity. Boolean retrieval is very effective if a query requires an exhaustive and unambiguous selection.
- The Boolean method offers a multitude of techniques to broaden or narrow a query.
- The Boolean approach can be especially effective in the later stages of the search process, because of the clarity and exactness with which relationships between concepts can be represented.

The standard Boolean approach has the following shortcomings:

- Users find it difficult to construct effective Boolean queries for several reasons. Users are using the natural language terms AND, OR or NOT that have a different meaning when used in a query. Thus, users will make errors when they form a Boolean query, because they resort to their knowledge of English. For example, in ordinary conversation a noun phrase of the form "A and B" usually refers to more entities than would "A" alone, whereas when used in the context of information retrieval it refers to fewer documents than would be retrieved by "A" alone. Hence, one of the common mistakes made by users is to substitute the AND logical operator for the OR logical operator when translating an English sentence to a Boolean query. Furthermore, to form complex queries, users must be familiar with the rules of precedence and the use of parentheses.
- Novice users have difficulty using parentheses, especially nested parentheses. Finally, users are overwhelmed by the multitude of ways a query can be structured or modified, because of the combinatorial explosion of feasible queries as the number of concepts increases. In particular, users have difficulty identifying and applying the different strategies that are available for narrowing or broadening a Boolean query.
- Only documents that satisfy a query exactly are retrieved. On the one hand, the AND operator is too severe because it does not distinguish between the case when none of the concepts are satisfied and the case where all except one are satisfied. Hence, no or very few documents are retrieved when more than three and four criteria are combined with the Boolean operator AND (referred to as the Null Output problem). On the other hand, the OR operator does not reflect how many concepts have been satisfied. Hence, often too many documents are retrieved (the Output Overload problem).

- It is difficult to control the number of retrieved documents. Users are often faced with the null-output or the information overload problem and they are at loss of how to modify the query to retrieve the reasonable number documents.
- The traditional Boolean approach does not provide a relevance ranking of the retrieved documents, although modern Boolean approaches can make use of the degree of coordination, field level and degree of stemming present to rank them.
- It does not represent the degree of uncertainty or error due the vocabulary problem.

1.3.1 Extended Boolean Model

To overcome the limitations of basic Boolean model, Salton, Fox and Wu introduced in 1983 the Extended Boolean Information Retrieval Model. Essentially the extended model

- Consider all of the terms in a query.
- Adjust the strictness of each AND or OR query operator with a p-value.
- Proposes a general model, p-norm that has as special cases the standard Boolean model (with fuzzy set interpretation --- when p is infinity) and the vector-space model (with inner-product similarity --- when p is one).
- Gets a spectrum of models with decreasing strictness, i.e., strict AND ... soft AND ... vector ... soft OR ... strict OR:
    - p-norm AND with p=infinity behaves like strict Boolean AND (i.e., MIN)
    - p-norm AND with p at moderate values softens the strictness of the AND
    - p-norm AND with p=1 behaves like p-norm OR with p=1 and behaves like vector space model
    - p-norm OR with p at moderate values softens the strictness of the OR
    - p-norm OR with p=infinity behaves like strict Boolean OR (i.e., MAX)
- Uses L-p family of norms to compute similarity by measuring:
    - distance from 0 point (i.e., none of query terms present) for OR;
    - 1 - distance from 1 point (i.e., all of query terms present) for AND.

The **P-norm** method developed by Fox (1983) allows query and document terms to have weights, which have been computed by using term frequency statistics with the proper normalization procedures. These normalized weights can be used to rank the documents in the order of decreasing distance from the point (0, 0, ... , 0) for an OR query, and in order of increasing distance from the point (1, 1, ... , 1) for an AND query. Further, the Boolean

operators have a coefficient P associated with them to indicate the degree of strictness of the operator (from 1 for least strict to infinity for most strict, i.e., the Boolean case). The P-norm uses a distance-based measure and the coefficient P determines the degree of exponentiation to be used. The exponentiation is an expensive computation, especially for P-values greater than one.

In **Fuzzy Set theory**, an element has a varying degree of membership to a set instead of the traditional binary membership choice. The weight of an index term for a given document reflects the degree to which this term describes the content of a document. Hence, this weight reflects the degree of membership of the document in the fuzzy set associated with the term in question. The degree of membership for union and intersection of two fuzzy sets is equal to the maximum and minimum, respectively, of the degrees of membership of the elements of the two sets. In the "Mixed Min and Max" model, the Boolean operators are softened by considering the query-document similarity to be a linear combination of the min and max weights of the documents.

1.4 INVERTED INDEX

An inverted index, also known as an inverted file, is a data structure central to text-based information retrieval. The name is derived from the structure's design and purpose, which in its simplest form is a map of key-value pairs:

- **Key:** The map is keyed by tokens, which can be a word, such as "cat" or "plate", or some other code (perhaps a part of a word) depending on the granularity of the index.
- **Value:** The value in the map is a list of postings, sometimes stored as a separate file on the file space and called a Postings File.

The inverted index is the output of the indexing process. The input to this process is a collection of documents of texts, often referred to in IR terms as a corpus. An inverted index is able to do many accesses in O(1) time at a price of significantly longer time to do an update, in the worst case O(n). Index construction time is longer as well, but query time is generally faster than with a b-tree. Since index construction is an off- line process, shorter query processing times at the expense of lengthier index construction times is an appropriate trade off.

Finally, inverted index storage structures can exceed the storage demands of the document collection itself. However, for many systems, the inverted index can be compressed to around ten percent of the original document collection. Given the alternative (of twenty minute

searches), search engine developers are happy to trade index construction time and storage for query efficiency. An inverted index is an optimized structure that is built primarily for retrieval, with update being only a secondary consideration. The basic structure inverts the text so that instead of the view obtained from scanning documents where a document is found and then its terms are seen (think of a list of documents each pointing to a list of terms it contains), an index is built that maps terms to documents (pretty much like the index found in the back of this book that maps terms to page numbers). Instead of listing each document once (and each term repeated for each document that contains the term), an inverted index lists each term in the collection only once and then shows a list of all the documents that contain the given term. Each document identifier is repeated for each term that is found in the document.

Example-1: For example, given the following documents:

Document 1: `The cat sat on the mat.`
Document 2: `The quick brown fox jumps over the lazy dog.`

Indexing documents 1 and 2 without employing any stop word removal or stemming would produce the following index…

```
{brown} -> {D2}
{cat}   -> {D1}
{dog}   -> {D2}
{fox}   -> {D2}
{jumps} -> {D2}
{mat}   -> {D1}
{on}    -> {D1}
{over}  -> {D2}
{quick} -> {D2}
{sat}   -> {D1}
{the}   -> {D1,D2}
```

This index only stores within the posting list the ID of the document a token is present in. It does not indicate how often a token occurs within a document, where it appears within a document or how long a document is (for normalisation purposes).

None-the-less, even an index of such simple form can answer the question “which documents contain the words ‘quick’ ‘dog’ and ‘the’?” It does not, however, answer (or support answering) the real question we want to ask, “Which documents are **about** <subject>?” very accurately.

Example-2: Consider the following two documents:

D1: The GDP increased 2 percent this quarter.

D2: The spring economic slowdown continued to spring downwards this quarter.

An inverted index for these two documents is given below:

2 → [D1]

continued → [D2]

downwards → [D2]

economic → [D2]

GDP → [D1]

increased → [D1]

percent → [D1]

quarter → [D1] → [D2]

slowdown → [D2]

spring → [D2]

the → [D1] → [D2]

this → [D1] → [D2]

to → [D2]

As shown, the terms *continued*, *economic*, *slowdown*, *spring*, and *to* appear in only the second document, the terms *GDP*, *increased*, and *percent*, and the numeral *2* appear in only the first document, and the terms *quarter*, *the*, and *this* appear in both documents.

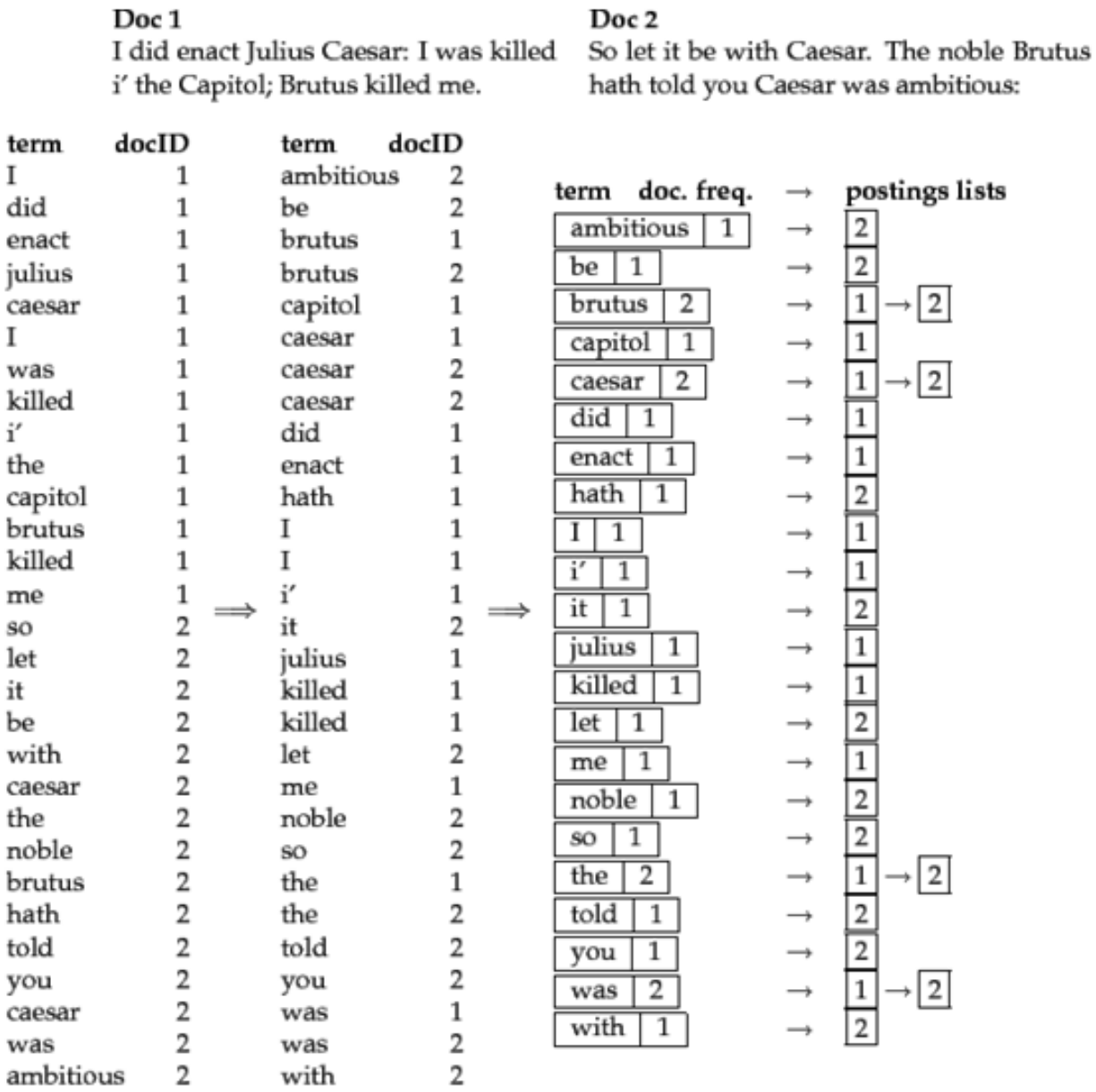
1.4.1 Steps in building an Inverted Index

The major steps in this are:

1. Collect the documents to be indexed:
   Friends, Romans, countrymen. | So let it be with Caesar ...
2. Tokenize the text, turning each document into a list of tokens:
   Friends | Romans | countrymen | So ...
3. Do linguistic pre-processing, producing a list of normalized tokens, which are the indexing terms:
   friend | roman | countryman | so ...
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Illustration:

Here, we assume that the first 3 steps have already been done, and we examine building a basic inverted index by *sort-based indexing*.



Here, within a document collection we assume that each document has a unique serial number, known as the document identifier (*docID*). During index construction, we can simply assign successive integers to each new document when it is first encountered. The input to indexing is a list of normalized tokens for each document, which we can equally think of as a list of pairs of term and docID, as shown above. The core indexing step is *sorting* this list so that the terms are alphabetical, giving us the representation in the middle column of the above shown table. Multiple occurrences of the same term from the same

document are then merged. The instances of the same term are then grouped, and the result is split into a *dictionary* and *postings* , as shown in the right column of table. Since a term generally occurs in a number of documents, this data organization already reduces the storage requirements of the index. The dictionary also records some statistics, such as the number of documents which contain each term (the *document frequency*, which is here also the length of each postings list). This information is not vital for a basic Boolean search engine, but it allows us to improve the efficiency of the search engine at query time, and it is a statistic later used in many ranked retrieval models. The postings are secondarily sorted by docID. This provides the basis for efficient query processing. This inverted index structure is essentially without rivals as the most efficient structure for supporting ad hoc text search.

In the resulting index, we pay for storage of both the dictionary and the postings lists. The latter are much larger, but the dictionary is commonly kept in memory, while postings lists are normally kept on disk, so the size of each is important.

What data structure should be used for a postings list?

A fixed length array would be wasteful as some words occur in many documents, and others in very few. For an in-memory postings list, two good alternatives are singly linked lists or variable length arrays. Singly linked lists allow cheap insertion of documents into postings lists (following updates, such as when re-crawling the web for updated documents), and naturally extend to more advanced indexing strategies such as skip lists, which require additional pointers. Variable length arrays win in space requirements by avoiding the overhead for pointers and in time requirements because their use of contiguous memory increases speed on modern processors with memory caches. Extra pointers can in practice be encoded into the lists as offsets. If updates are relatively infrequent, variable length arrays will be more compact and faster to traverse. We can also use a hybrid scheme with a linked list of fixed length arrays for each term. When postings lists are stored on disk, they are stored as a contiguous run of postings without explicit pointers so as to minimize the size of the postings list and the number of disk seeks to read a postings list into memory.

1.4.2 Applications

The inverted index data structure is a central component of a typical search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs. Once a forward index is developed, which stores lists of words per document; it is next inverted to develop an inverted index. Querying the

forward index would require sequential iteration through each document and to each word to verify a matching document. The time, memory, and processing resources to perform such a query are not always technically realistic. Instead of listing the words per document in the forward index, the inverted index data structure is developed which lists the documents per word.

With the inverted index created, the query can now be resolved by jumping to the word id (via random access) in the inverted index.In pre-computer times, concordances to important books were manually assembled. These were effectively inverted indexes with a small amount of accompanying commentary that required a tremendous amount of effort to produce.

In bioinformatics, inverted indexes are very important in the sequence assembly of short fragments of sequenced DNA. One way to find the source of a fragment is to search for it against a reference DNA sequence. A small number of mismatches (due to differences between the sequenced DNA and reference DNA, or errors) can be accounted for by dividing the fragment into smaller fragments—at least one sub-fragment is likely to match the reference DNA sequence. The matching requires constructing an inverted index of all substrings of a certain length from the reference DNA sequence. Since the human DNA contains more than 3 billion base pairs, and we need to store a DNA substring for every index, and a 32-bit integer for index itself, the storage requirement for such an inverted index would probably be in the tens of gigabytes, just beyond the available RAM capacity of most personal computers today.

1.5 SUMMARY

In this unit, we have described the role of information system for present day needs and its importance. The development of information system is presented in detail. The design of information retrieval system using Boolean model along with its extended model which exists to overcome the limitations of the basic Boolean model. The design of inverted index used in information retrieval system is discussed with many examples. At the end, we have given some applications of IR system.

1.6 KEYWORDS

Information retrieval, Boolean Model, P-norm, Fuzzy theory, Inverted index.

1.7 QUESTIONS

1. Define information retrieval and explain its importance.
2. Give the brief history of information system.
3. Describe the information retrieval model from mathematical perception.
4. Describe the information retrieval model from properties perception.
5. Discuss the information retrieval system based on Boolean model.
6. What are the merits and demerits of Boolean model?
7. Explain Extended Boolean model.
8. With an example, explain how inverted index is developed.
9. What are the applications of IR?
10. Draw the inverted index that would be built for the following document collection.

    Doc 1 new home sales top forecasts

    Doc 2 home sales rise in July

    Doc 3 increase in home sales in July

    Doc 4 July new home sales rise

11. Consider these documents:

    Doc 1 breakthrough drug for schizophrenia

    Doc 2 new schizophrenia drug

    Doc 3 new approach for treatment of schizophrenia

    Doc 4 new hopes for schizophrenia patients

    i. Draw the term-document incidence matrix for this document collection.

    ii. Draw the inverted index representation for this collection.

12. For the document collection illustrated in inverted index construction, what are the returned results for these queries:

    i. schizophrenia AND drug

    ii. for AND NOT(drug OR approach)

1.8 REFERENCES FOR FURTHER READING/STUDIES

Singhal, Amit (2001). "Modern Information Retrieval: A Brief Overview". Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35–43.

Korfhage, Robert R. (1997). Information Storage and Retrieval. Wiley. pp. 368 pp.. ISBN 978-0-471-14338-3.

Frakes, William B. (1992). Information Retrieval Data Structures & Algorithms. Prentice-Hall, Inc.. ISBN 0-13-463837-9.

Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich (2008). Introduction to Information Retrieval. Cambridge University Press.

G. Salton, E. Fox, and H. Wu. Extended Boolean Information Retrieval. Communications of the ACM, 1983, 26(11): 1022-1036.

Justin Zobel and Alistair Moffat, Inverted Files for Text Search Engines, ACM Computing Surveys, 38(2), article 6, July 2006.

UNIT 2: FUNDAMENTALS OF INFORMATION RETRIEVAL SYSTEMS

Structure

2.0 INTRODUCTION

In this unit, we introduce and define basic IR concepts, and present a domain model of IR systems that describes their similarities and differences. The domain model is used to introduce and relate the chapters that follow. The relationship of IR systems to other information systems is discussed, as is the evaluation of IR systems.

Automated information retrieval (IR) systems were originally developed to help manage the huge scientific literature that has developed since the 1940s. Many university, corporate, and public libraries now use IR systems to provide access to books, journals, and other documents. Commercial IR systems offer databases containing millions of documents in myriad subject areas. Dictionary and encyclopedia databases are now widely available for PCs. IR has been found useful in such disparate areas as office automation and software engineering. Indeed, any discipline that relies on documents to do its work could potentially use and benefit from IR.

An IR system matches user *queries*--formal statements of information needs--to documents stored in a database. A document is a data object, usually textual, though it may also contain other types of data such as photographs, graphs, and so on. Often, the documents themselves are not stored directly in the IR system, but are represented in the system by document surrogates. This unit, for example, is a document and could be stored in its entirety in an IR database. One might instead, however, choose to create a document surrogate for it consisting of the title, author, and abstract. This is typically done for efficiency, that is, to reduce the size of the database and searching time.

An IR system must support certain basic operations. There must be a way to enter documents into a database, change the documents, and delete them. There must also be some way to search for documents, and present them to a user. As the following sections illustrate, IR systems vary greatly in the ways they accomplish these tasks. In the next section, the similarities and differences among IR systems are discussed.

2.1 ANALYSIS OF IR SYSTEMS

In order to find, understand, and use algorithms and associated data structures effectively, it is necessary to have a conceptual framework. Analysis attempts to discover and record the similarities and differences among related systems based on the domain of information.

The first steps in domain analysis are to identify important concepts and vocabulary in the domain, define them, and organize them with a faceted classification. Table 3.1 is a faceted classification for IR systems, containing important IR concepts and vocabulary. The first row of the table specifies the facets--that is, the attributes that IR systems share. Facets represent the parts of IR systems that will tend to be constant from system to system. For example, all IR systems must have a database structure--they vary in the database structures they have; some have inverted file structures, some have flat file structures, and so on.

A given IR system can be classified by the facets and facet values, called terms that it has. Terms within a facet are not mutually exclusive, and more than one term from a facet can be used for a given system. Some decisions constrain others. If one chooses a Boolean conceptual model, for example, then one must choose a parse method for queries.

Viewed another way, each facet is a design decision point in developing the architecture for an IR system. The system designer must choose, for each facet, from the alternative terms for that facet. We will now discuss the facets and their terms in greater detail.

Table2.1: Faceted Classification of IR Systems

Conceptual Model	File Structure	Query	Term Operations	Document Operations	Hardware Operations
Boolean	Flat File	Feedback	Stem	Parse	vonNeumann
Extended	Inverted	Parse	Weight	Display	Parallel
Boolean	File				
Probabilistic	Signature	Boolean	Thesaurus	Cluster	
String Search	Pat Trees	Cluster	Stoplist	Rank	Optical Disk
Vector Space	Graphs		Truncation	Sort	Mag. Disk

2.2 CONCEPTUAL MODELS OF IR

The most general facet in the previous classification scheme is *conceptual model*. An IR conceptual model is a general approach to IR systems. Several taxonomies for IR conceptual models have been proposed. Faloutsos (1985) gives three basic approaches: text pattern search, inverted file search, and signature search. Belkin and Croft (1987) categorize IR conceptual models differently. They divide retrieval techniques first into exact match and inexact match. The exact match category contains text pattern search and Boolean search techniques. The inexact match category contains such techniques as probabilistic, vector space, and clustering, among others. The problem with these taxonomies is that the categories are not mutually exclusive, and a single system may contain aspects of many of them.

Almost all of the IR systems fielded today are either Boolean IR systems or text pattern search systems. Text pattern search queries are strings or regular expressions. Text pattern systems are more common for searching small collections, such as personal collections of files. The grep family of tools, described in Earhart (1986), in the UNIX environment is a well-known example of text pattern searchers.

Almost all of the IR systems for searching large document collections are Boolean systems. In a Boolean IR system, documents are represented by sets of keywords, usually stored in an inverted file. An inverted file is a list of keywords and identifiers of the documents in which

they occur. Boolean queries are keywords connected with Boolean logical operators (AND, OR, NOT).

Researchers have also tried to improve IR performance by using information about the statistical distribution of terms that is the frequencies with which terms occur in documents, document collections, or subsets of document collections such as documents considered relevant to a query. Term distributions are exploited within the context of some statistical model such as the vector space model, the probabilistic model, or the clustering model. Using these probabilistic models and information about term distributions, it is possible to assign a probability of relevance to each document in a retrieved set allowing retrieved documents to be ranked in order of probable relevance. Ranking is useful because of the large document sets that are often retrieved. In addition to the ranking algorithms, it is possible to group (cluster) documents based on the terms that they contain and to retrieve from these groups using a ranking methodology. Methods for clustering documents and retrieving from these clusters are discussed later.

2.3 FILE STRUCTURES

A fundamental decision in the design of IR systems is which type of file structure to use for the underlying document database. As can be seen in Table 3.1, the file structures used in IR systems are flat files, inverted files, signature files, PAT trees, and graphs. Though it is possible to keep file structures in main memory, in practice IR databases are usually stored on disk because of their size.

Using a flat file approach, one or more documents are stored in a file, usually as ASCII or EBCDIC text. Flat file searching is usually done via pattern matching. On UNIX, for example, one can store a document collection one per file in a UNIX directory, and search it using pattern searching tools such as grep (Earhart 1986) or awk (Aho, Kernighan, and Weinberger 1988).

An inverted file is a kind of indexed file. The structure of an inverted file entry is usually keyword, document-ID, field-ID. A keyword is an indexing term that describes the document, document-ID is a unique identifier for a document, and field-ID is a unique name that indicates from which field in the document the keyword came. Some systems also include

information about the paragraph and sentence location where the term occurs. Searching is done by looking up query terms in the inverted file.

Signature files contain signatures--it patterns--that represent documents. There are various ways of constructing signatures. Using one common signature method, for example, documents are split into logical blocks each containing a fixed number of distinct significant, that is, non-stop list words. Each word in the block is hashed to give a signature--a bit pattern with some of the bits set to 1. The signatures of each word in a block are OR'ed together to create a block signature. The block signatures are then concatenated to produce the document signature. Searching is done by comparing the signatures of queries with document signatures.

PAT trees are Patricia trees constructed over all *sistrings* in a text. If a document collection is viewed as a sequentially numbered array of characters, a sistring is a subsequence of characters from the array starting at a given point and extending an arbitrary distance to the right. A Patricia tree is a digital tree where the individual bits of the keys are used to decide branching.

Graphs, or networks, are ordered collections of nodes connected by arcs. They can be used to represent documents in various ways. For example, a kind of graph called a semantic net can be used to represent the semantic relationships in text often lost in the indexing systems above. Although interesting, graph-based techniques for IR are impractical now because of the amount of manual effort that would be needed to represent a large document collection in this form. Since graph-based approaches are currently impractical, we have not covered them in detail in this book.

2.4 QUERY OPERATIONS

Queries are formal statements of information needs put to the IR system by users. The operations on queries are obviously a function of the type of query, and the capabilities of the IR system. One common query operation is parsing, that is breaking the query into its constituent elements. Boolean queries, for example, must be parsed into their constituent terms and operators. The set of document identifiers associated with each query term is retrieved, and the sets are then combined according to the Boolean operators.

In feedback, information from previous searches is used to modify queries. For example, terms from relevant documents found by a query may be added to the query, and terms from non-relevant documents deleted. There is some evidence that feedback can significantly improve IR performance.

2.5 TERM OPERATIONS

Operations on terms in an IR system include stemming, truncation, weighting, and stop-list and thesaurus operations. Stemming is the automated conflation (fusing or combining) of related words, usually by reducing the words to a common root form. Truncation is manual conflation of terms by using wildcard characters in the word, so that the truncated term will match multiple words. For example, a searcher interested in finding documents about truncation might enter the term "truncat?" which would match terms such as truncate, truncated, and truncation. Another way of conflating related terms is with a thesaurus which lists synonymous terms, and sometimes the relationships among them. A stop-list is a list of words considered to have no indexing value, used to eliminate potential indexing terms. Each potential indexing term is checked against the stop-list and eliminated if found there.

In term weighting, indexing or query terms are assigned numerical values usually based on information about the statistical distribution of terms, that is, the frequencies with which terms occur in documents, document collections, or subsets of document collections such as documents considered relevant to a query.

2.6 DOCUMENT OPERATIONS

Documents are the primary objects in IR systems and there are many operations for them. In many types of IR systems, documents added to a database must be given unique identifiers, parsed into their constituent fields, and those fields broken into field identifiers and terms. Once in the database, one sometimes wishes to mask off certain fields for searching and display. For example, the searcher may wish to search only the title and abstract fields of documents for a given query, or may wish to see only the title and author of retrieved documents. One may also wish to sort retrieved documents by some field, for example by author. Display operations include printing the documents, and displaying them on a CRT.

Using information about term distributions, it is possible to assign a probability of relevance to each document in a retrieved set, allowing retrieved documents to be ranked in order of

probable relevance. Term distribution information can also be used to cluster similar documents in a document space.

Another important document operation is display. The user interface of an IR system, as with any other type of information system, is critical to its successful usage.

2.7 HARDWARE FOR IR

Hardware affects the design of IR systems because it determines, in part, the operating speed of an IR system--a crucial factor in interactive information systems--and the amounts and types of information that can be stored practically in an IR system. Most IR systems in use today are implemented on von Neumann machines--general purpose computers with a single processor. The computing speeds of these machines have improved enormously over the years, yet there is still IR applications for which they may be too slow. In response to this problem, some researchers have examined alternative hardware for implementing IR systems. There are two approaches--parallel computers and IR specific hardware.

Along with the need for greater speed has come the need for storage media capable of compactly holding the huge document databases that have proliferated. Optical storage technology, capable of holding gigabytes of information on a single disk, has met this need.

2.8 FUNCTIONAL VIEW OF PARADIGM IR SYSTEM

Figure 2.1 shows the activities associated with a common type of Boolean IR system, chosen because it represents the operational standard for IR systems.

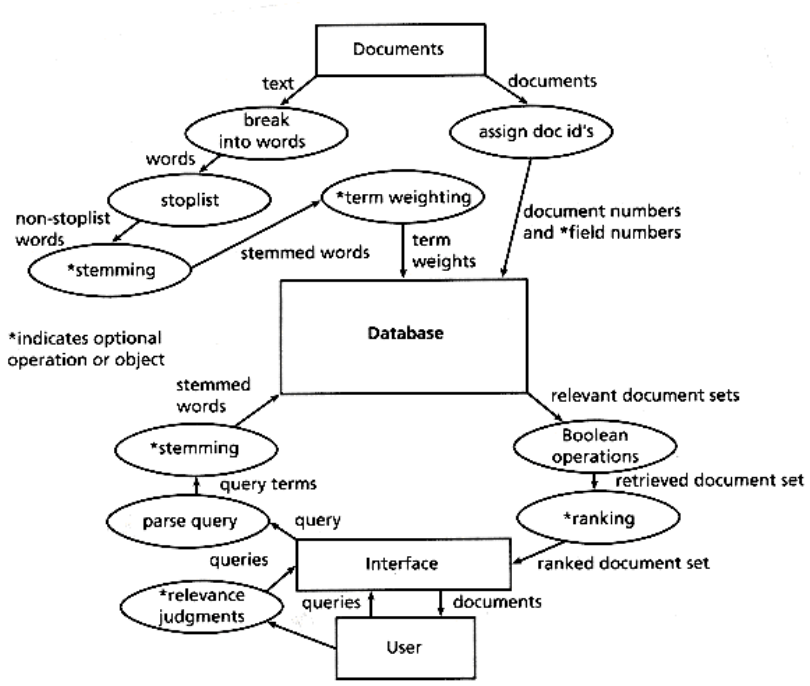


Figure 2.1: Example of Boolean IR system

When building the database, documents are taken one by one, and their text is broken into words. The words from the documents are compared against a stop-list--a list of words thought to have no indexing value. Words from the document not found in the stop-list may next be stemmed. Words may then also be counted, since the frequency of words in documents and in the database as a whole are often used for ranking retrieved documents. Finally, the words and associated information such as the documents, fields within the documents, and counts are put into the database. The database then might consist of pairs of document identifiers and keywords as follows.

```
keyword1 - document1-Field_2
keyword2 - document1-Field_2, 5
keyword2 - document3-Field_1, 2
keyword3 - document3-Field_3, 4
◆
◆
◆
keyword-n - document-n-Field_i, j
```

Such a structure is called an *inverted file.* In an IR system, each document must have a unique identifier, and its fields, if field operations are supported, must have unique field names.

To search the database, a user enters a query consisting of a set of keywords connected by Boolean operators (AND, OR, NOT). The query is parsed into its constituent terms and Boolean operators. These terms are then looked up in the inverted file and the list of

document identifiers corresponding to them are combined according to the specified Boolean operators. If frequency information has been kept, the retrieved set may be ranked in order of probable relevance. The result of the search is then presented to the user. In some systems, the user makes judgments about the relevance of the retrieved documents, and this information is used to modify the query automatically by adding terms from relevant documents and deleting terms from non-relevant documents. Systems such as this give remarkably good retrieval performance given their simplicity, but their performance are far from perfect.

2.9 IR AND OTHER TYPES OF INFORMATION SYSTEMS

How do IR systems relate to different types of information systems such as database management systems (DBMS), and artificial intelligence (AI) systems? Table 2.3 summarizes some of the similarities and differences.

Table 2.3: IR, DBMS, Al Comparison

	Data Object	**Primary Operation**	**Database Size**
IR	document	retrieval (probabilistic)	small to very large
DBMS	table	retrieval (deterministic)	small to very large
AI	logical statements	inference	usually small

One difference between IR, DBMS, and AI systems is the amount of usable structure in their data objects. Documents, being primarily text, in general have less usable structure than the tables of data used by relational DBMS, and structures such as frames and semantic nets used by AI systems. It is possible, of course, to analyze a document manually and store information about its syntax and semantics in a DBMS or an AI system. The barriers for doing this to a large collection of documents are practical rather than theoretical. The work involved in doing knowledge engineering on a set of say 50,000 documents would be enormous. Researchers have devoted much effort to constructing hybrid systems using IR, DBMS, AI, and other techniques; see, for example, Tong (1989). The hope is to eventually develop practical systems that combine IR, DBMS, and AI.

Another distinguishing feature of IR systems is that retrieval is probabilistic. That is, one cannot be certain that a retrieved document will meet the information need of the user. In a typical search in an IR system, some relevant documents will be missed and some non-relevant documents will be retrieved. This may be contrasted with retrieval from, for example, a DBMS where retrieval is deterministic. In a DBMS, queries consist of attribute-value pairs that either match, or do not match, records in the database.

One feature of IR systems shared with many DBMS is that their databases are often very large--sometimes in the gigabyte range. Book library systems, for example, may contain several million records. Commercial on-line retrieval services such as Dialog and BRS provide databases of many gigabytes. The need to search such large collections in real time places severe demands on the systems used to search them. Selection of the best data structures and algorithms to build such systems is often critical.

Another feature that IR systems share with DBMS is database volatility. A typical large IR application, such as a book library system or commercial document retrieval service, will change constantly as documents are added, changed, and deleted. This constrains the kinds of data structures and algorithms that can be used for IR.

In summary, a typical IR system must meet the following functional and nonfunctional requirements. It must allow a user to add, delete, and change documents in the database. It must provide a way for users to search for documents by entering queries, and examine the retrieved documents. It must accommodate databases in the megabyte to gigabyte range, and retrieve relevant documents in response to queries interactively--often within 1 to 10 seconds.

2.10 IR SYSTEM EVALUATION

IR systems can be evaluated in terms of many criteria including execution efficiency, storage efficiency, retrieval effectiveness, and the features they offer a user. The relative importance of these factors must be decided by the designers of the system, and the selection of appropriate data structures and algorithms for implementation will depend on these decisions.

Execution efficiency is measured by the time it takes a system, or part of a system, to perform a computation. This can be measured in C based systems by using profiling tools such as prof (Earhart 1986) on UNIX. Execution efficiency has always been a major concern of IR systems since most of them are interactive, and a long retrieval time will interfere with the

usefulness of the system. The non-functional requirements of IR systems usually specify maximum acceptable times for searching, and for database maintenance operations such as adding and deleting documents.

Storage efficiency is measured by the number of bytes needed to store data. Space overhead, a common measure of storage efficiency, is the ratio of the size of the index files plus the size of the document files over the size of the document files. Space overhead ratios of from 1.5 to 3 are typical for IR systems based on inverted files.

Most IR experimentation has focused on retrieval effectiveness--usually based on document *relevance judgments*. This has been a problem since relevance judgments are subjective and unreliable. That is, different judges will assign different relevance values to a document retrieved in response to a given query. The seriousness of the problem is the subject of debate, with many IR researchers arguing that the relevance judgment reliability problem is not sufficient to invalidate the experiments that use relevance judgments.

Many measures of retrieval effectiveness have been proposed. The most commonly used are *recall* and *precision*. Recall is the ratio of relevant documents retrieved for a given query over the number of relevant documents for that query in the database. Except for small test collections, this denominator is generally unknown and must be estimated by sampling or some other method. Precision is the ratio of the number of relevant documents retrieved over the total number of documents retrieved. Both recall and precision take on values between 0 and 1.

Since one often wishes to compare IR performance in terms of both recall and precision, methods for evaluating them simultaneously have been developed. One method involves the use of recall-precision graphs--bivariate plots where one axis is recall and the other precision. Figure 3.2 shows an example of such a plot. Recall-precision plots show that recall and precision are inversely related. That is, when precision goes up, recall typically goes down and vice-versa. Such plots can be done for individual queries, or averaged over queries as described in Salton and McGill (1983), and van Rijsbergen (1979).

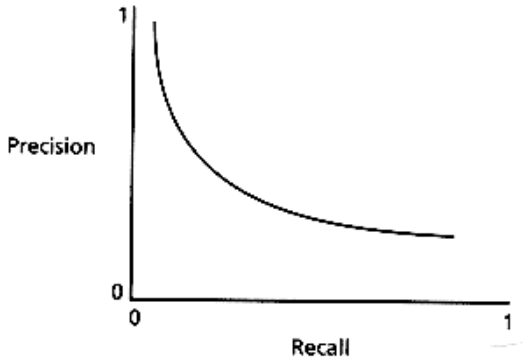


Figure 3.2: Recall-precision graph

A combined measure of recall and precision, E, has been developed by van Rijsbergen (1979). The evaluation measure *E* is defined as:

$$E = 1 - \frac{(1 + b^2) P R}{b^2 P + R}$$

where *P* = precision, *R* = recall, and *b* is a measure of the relative importance, to a user, of recall and precision. Experimenters choose values of *E* that they hope will reflect the recall and precision interests of the typical user. For example, *b* levels of .5, indicating that a user was twice as interested in precision as recall, and 2, indicating that a user was twice as interested in recall as precision, might be used.

IR experiments often use test collections which consist of a document database and a set of queries for the data base for which relevance judgments are available. The number of documents in test collections has tended to be small, typically a few hundred to a few thousand documents.

2.11 SUMMARY

This unit introduced and defined basic IR concepts, and presented a domain model of IR systems that describes their similarities and differences. A typical IR system must meet the following functional and non-functional requirements. It must allow a user to add, delete, and change documents in the database. It must provide a way for users to search for documents by entering queries, and examine the retrieved documents. An IR system will typically need to support large databases, some in the megabyte to gigabyte range, and retrieve relevant documents in response to queries interactively--often within 1 to 10 seconds. We have

summarized the various approaches, elaborated in subsequent chapters, taken by IR systems in providing these services. Evaluation techniques for IR systems were also briefly surveyed.

2.12 KEYWORDS

Information retrieval systems, Conceptual model, Term operations, Document operations, Evaluation

2.13 QUESTIONS

1. Give the classification of IR systems
2. Explain the conceptual model of IR system.
3. Write a note on file structures
4. What are query operations?
5. What are term operations?
6. What are document operations?
7. With the block diagram, explain the functional overview of IR system.
8. Compare IR system to other information management systems.

2.14 REFERENCES FOR FURTHER READING/STUDIES

Aho, A., B. Kernighan, and P. Weinberger. 1988. *The AWK Programming Language.* Reading, Mass.: Addison-Wesley.

Belkin N. J., and W. B. Croft. 1987. "Retrieval Techniques," in *Annual Review of Information Science and Technology*, ed. M. Williams. New York: Elsevier Science Publishers, 109-145.

Earhart, S. 1986. *The UNIX Programming Language*, vol. 1. New York: Holt, Rinehart, and Winston.

Faloutsos, C. 1985. "Access Methods for Text," *Computing Surveys*, 17(1), 49-74.

Fox, E., ed. 1990. Virginia Disk One, Blacksburg: Virginia Polytechnic Institute and State University.

Frakes, W. B. 1984. "Term Conflation for Information Retrieval," in *Research and Development in Information Retrieval*, ed. C. S. van Rijsbergen. Cambridge: Cambridge University Press.

Prieto-Diaz, R., and G. Arango. 1991. *Domain Analysis: Acquisition of Reusable Information for Software Construction*. New York: IEEE Press.

Salton, G., and M. Mcgill 1983. *An Introduction to Modern Information Retrieval*. New York: McGraw-Hill.

Sedgewick, R. 1990. *Algorithms in C*. Reading, Mass.: Addison-Wesley.

Sparck-Jones, K. 1981. *Information Retrieval Experiment*. London: Butterworths.

Tong, R, ed. 1989. Special Issue on Knowledge Based Techniques for Information Retrieval, *International Journal of Intelligent Systems*, 4(3).

Van Rijsbergen, C. J. 1979. *Information Retrieval*. London: Butterworths.

UNIT 3: BASICS OF INFORMATION RETRIEVAL SYSTEM

Structure

3.0 INTRODUCTION

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers). In simple terms, Information Retrieval (IR) deals with the representation, storage and organization of unstructured data. Information retrieval is the process of searching within a document collection for a particular information need (a query). Its mission is to assist in information search.

An Information Retrieval System is a system that is capable of storage, retrieval, and maintenance of information. Information in this context can be composed of text (including numeric and date data), images, audio, video and other multi-media objects. Although the form of an object in an Information Retrieval System is diverse, the text aspect has been the only data type that lent itself to fully functional processing. The other data types have been treated as highly informative sources, but are primarily linked for retrieval based upon search of the text. Techniques are beginning to emerge to search these other media types. Commercial development of pattern matching against other data types is starting to be a common function integrated within the total information system. In some systems the text may only be an identifier to display another associated data type that holds the substantive

information desired by the system's users (e.g., using closed captioning to locate video of interest.)

The term "item" is used to represent the smallest complete unit that is processed and manipulated by the system. The definition of item varies by how a specific source treats information. A complete document, such as a book, newspaper or magazine could be an item. At other times each chapter or article may be defined as an item. As sources vary and systems include more complex processing, an item may address even lower levels of abstraction such as a contiguous passage of text or a paragraph. For readability, throughout this book the terms "item" and "document" are not in this rigorous definition, but used most of the book it is best to consider an item as text. But in reality an item may be a combination of many modals of information. For example a video news program could be considered an item. It is composed of text in the form of closed captioning, audio text provided by the speakers, and the video images being displayed. There are multiple "tracks" of information possible in a single item. They are typically correlated by time. Where the text discusses multimedia information retrieval keep this expanded model in mind.

An Information Retrieval System consists of a software program that facilitates a user in finding the information the user needs. The system may use standard computer hardware or specialized hardware to support the search sub function and to convert non-textual sources to a searchable media (e.g., transcription of audio to text). The gauge of success of an information system is how well it can minimize the overhead for a user to find the needed information. Overhead from a user's perspective is the time required to find the information needed, excluding the time for actually reading the relevant data. Thus search composition, search execution, and reading non-relevant items are all aspects of information retrieval overhead.

The first Information Retrieval Systems originated with the need to organize information in central repositories (e.g., libraries). Catalogues were created to facilitate the identification and retrieval of items. Original definitions focused on "documents" for information retrieval (or their surrogates) rather than the multi-media integrated information that is now available. As computers became commercially available, they were obvious candidates for the storage and retrieval of text. Early introduction of Database Management Systems provided an ideal platform for electronic manipulation of the indexes to information. Libraries followed the paradigm of their catalogs and references by migrating the format and organization of their hardcopy information references into structured databases. These remain as a primary

mechanism for researching sources of needed information and play a major role in available Information Retrieval Systems. Academic research that was pursued through the 1980s was constrained by the paradigm of the indexed structure associated with libraries and the lack of computer power to handle large (gigabyte) text databases. The Military and other Government entities have always had a requirement to store and search large textual databases. As a result they began many independent developments of textual Information Retrieval Systems. Given the large quantities of data they needed to process, they pursued both research and development of specialized hardware and unique software solutions incorporating Commercial off the Shelf (COTS) products where possible. The Government has been the major funding source of research into Information Retrieval Systems.

With the advent of inexpensive powerful personnel computer processing systems and high speed, large capacity secondary storage products, it has become commercially feasible to provide large textual information databases for the average user. The introduction and exponential growth of the Internet along with its initial WAIS (Wide Area Information Servers) capability and more recently advanced search servers (e.g., INFOSEEK, EXCITE) has provided a new avenue for access to terabytes of information. The algorithms and techniques to optimize the processing and access of large quantities of textual data were once the sole domain of segments of the Government, a few industries, and academics. They have now become a needed capability for large quantities of the population with significant research and development being done by the private sector. Additionally the volumes of non-textual information are also becoming searchable using specialized search capabilities. News organizations such as the BBC are processing the audio news they have produced and are making historical audio news searchable via the audio transcribed versions of the news. Major video organizations such as Disney are using video indexing to assist in finding specific images in their previously produced videos to use in future videos or incorporate in advertising. With exponential growth of multi-media on the Internet capabilities such as these are becoming common place. Information Retrieval exploitation of multi-media is still in its infancy with significant theoretical and practical knowledge missing.

3.1 DATA RETRIEVAL SYSTEMS V/s INFORMATION RETRIEVAL SYSTEMS

With the rapid growth of information and easy access of information, in particular the boom of the World Wide Web, the problem of finding useful information and knowledge becomes one of the most important topics in information and computer science. Web browsers, Web

search engines designed based on the theory of information retrieval (IR), and information retrieval systems (IRS) are some of the solutions to this problem. They aim at providing a user with useful and relevant information in response to a user query. IRS, Web browsers, and Web search engines extend the basic search functionalities of data retrieval systems (DRS) exemplified by a database system. A major difference between data retrieval (DR) and IR lies in the nature of their problem domains [10]. DR deals with well defined, structured and simple problems, where data items, queries, and matching methods can be precisely defined and interpreted. In contrast, IR deals with not-so-well defined, semi-structured or un-structured, and more complicated problems, where infor mation items (documents), user information needs and queries, and matching methods cannot be precisely defined. In an DRS, a user can only perform a well structured task of search. In other word, a user needs to supply a query and the system provides results based on an exact match of data items and the query. With an IRS, a user can perform less structured tasks. The evolution from DRS to IRS increases the power of a user in finding useful information. Current IRS, Web browsers, and Web search engines provide basic functionalities to assist a user in the context of libraries and in the early stage of the Web. When finding useful information, a user may need to perform more tasks, such as understanding, analysis, organization, and discovery, in addition to the conventional tasks of search and browsing. With the recent development of XML (eXtensible Markup Language), it is possible to express both the structure and semantics information about a document. A user can perform additional tasks with respect to an XML document collection. It is expected that current IRS need to be extended to support more user tasks. The next evolution of retrieval systems is to move from IRS to information retrieval support systems (IRSS). IRSS is based on a different design philosophy that emphasizes the supporting functionality of the system, instead of the specific search and browsing functionalities. In the process of finding useful information, a user plays an active role in an IRSS by using the utilities, tools, and languages provided by the system.

3.2 OBJECTIVES OF INFORMATION RETRIEVAL SYSTEMS

The general objective of an Information Retrieval System is to minimize the overhead of a user locating needed information. Overhead can be expressed as the time a user spends in all of the steps leading to reading an item containing the needed information (e.g., query generation, query execution, scanning results of query to select items to read, reading non-relevant items). The success of an information system is very subjective, based upon what information is needed and the willingness of a user to accept overhead. Under some

circumstances, needed information can be defined as all information that is in the system that relates to a user's need. In other cases it may be defined as sufficient information in the system to complete a task, allowing for missed data. For example, a financial advisor recommending a billion dollar purchase of another company needs to be sure that all relevant, significant information on the target company has been located and reviewed in writing the recommendation. In contrast, a student only requires sufficient references in a research paper to satisfy the expectations of the teacher, which never is all inclusive. A system that supports reasonable retrieval requires fewer features than one which requires comprehensive retrieval. In many cases comprehensive retrieval is a negative feature because it overloads the user with more information than is needed. This makes it more difficult for the user to filter the relevant but non-useful information from the critical items. In information retrieval the term "relevant" item is used to represent an item.

3.3 MEASURES OF INFORMATION RETRIEVAL SYSTEMS

Many different measures for evaluating the performance of information retrieval systems have been proposed. The measures require a collection of documents and a query. All common measures described here assume a ground truth notion of relevancy: every document is known to be either relevant or non-relevant to a particular query. In practice queries may be ill-posed and there may be different shades of relevancy. The two major measures commonly associated with information systems are precision and recall.

Precision: Precision is the fraction of the documents retrieved that are relevant to the user's information need.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

In binary classification, precision is analogous to positive predictive value. Precision takes all retrieved documents into account. It can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called *precision at n* or *P@n*.

Note that the meaning and usage of "precision" in the field of Information Retrieval differs from the definition of accuracy and precision within other branches of science and technology.

Recall: Recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

In binary classification, recall is often called sensitivity. So it can be looked at as *the probability that a relevant document is retrieved by the query*.

It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision.

Fall-out: The proportion of non-relevant documents that are retrieved, out of all non-relevant documents available:

$$\text{fall-out} = \frac{|\{\text{non-relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{non-relevant documents}\}|}$$

In binary classification, fall-out is closely related to specificity and is equal to (1-Specificity). It can be looked at as *the probability that a non-relevant document is retrieved by the query*. It is trivial to achieve fall-out of 0% by returning zero documents in response to any query.

F-measure: The weighted harmonic mean of precision and recall, the traditional F-measure or balanced F-score is:

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})}.$$

This is also known as the F_1 measure, because recall and precision are evenly weighted.

The general formula for non-negative real β is:

$$F_\beta = \frac{(1+\beta^2) \cdot (\text{precision} \cdot \text{recall})}{(\beta^2 \cdot \text{precision} + \text{recall})}.$$

Two other commonly used F measures are F_2 the measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

The F-measure was derived by van Rijsbergen (1979) so that F_β measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision". It is based on van Rijsbergen's effectiveness measure

$$E = 1 - \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}}.$$

Their relationship is $F_\beta = 1 - E$ where $\alpha = \frac{1}{1+\beta^2}$.

Average precision: Precision and recall are single-value metrics based on the whole list of documents returned by the system. For systems that return a ranked sequence of documents, it is desirable to also consider the order in which the returned documents are presented. By computing a precision and recall at every position in the ranked sequence of documents, one can plot a precision-recall curve, plotting precision *p(r)* as a function of recall *r*. Average precision computes the average value of *p(r)* over the interval from *r*=0 to *r*=1

$$\mathrm{AveP} = \int_0^1 p(r)dr.$$

This integral is in practice replaced with a finite sum over every position in the ranked sequence of documents:

$$\mathrm{AveP} = \sum_{k=1}^{n} P(k)\Delta r(k)$$

where *k* is the rank in the sequence of retrieved documents, *n* is the number of retrieved documents, *p(k)* is the precision at cut-off *k* in the list, and $\Delta r(k)$ is the change in recall from items *k-1* to *k.*

This finite sum is equivalent to:

$$\mathrm{AveP} = \frac{\sum_{k=1}^{n} (P(k) \times rel(k))}{\text{number of relevant documents}}$$

where *rel(k)* is an indicator function equalling 1 if the item at rank *k* is a relevant document, zero otherwise. Note that the average is over all relevant documents and the relevant documents not retrieved get a precision score of zero.

Some authors choose to interpolate the *p(r)* function to reduce the impact of "wiggles" in the curve. For example, the PASCAL Visual Object Classes challenge (a benchmark for computer vision object detection) computes average precision by averaging the precision over a set of evenly spaced recall levels {0, 0.1, 0.2, ... 1.0}.

$$\mathrm{AveP} = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1.0\}} p_{\mathrm{interp}}(r)$$

where $p_{interp}(r)$ is an interpolated precision that takes the maximum precision over all recalls greater than *r*:

$$p_{\mathrm{interp}}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}).$$

An alternative is to derive an analytical function *p(r)* by assuming a particular parametric distribution for the underlying decision values. For example, a *binormal precision-recall curve* can be obtained by assuming decision values in both classes to follow a Gaussian distribution.

Average precision is also sometimes referred to geometrically as the area under the precision-recall curve.

R-Precision: Precision at R-th position in the ranking of results for a query that has R relevant documents. This measure is highly correlated to Average Precision. Also, Precision is equal to Recall at the R-th position.

Mean average precision: Mean average precision for a set of queries is the mean of the average precision scores for each query.

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q}$$

Where *Q* is the number of queries.

Discounted cumulative gain: DCG uses a graded relevance scale of documents from the result set to evaluate the usefulness, or gain, of a document based on its position in the result list. The premise of DCG is that highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result.

The DCG accumulated at a particular rank position *p* is defined as:

$$\mathrm{DCG_p} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}.$$

Since result set may vary in size among different queries or systems, to compare performances the normalised version of DCG uses an ideal DCG. To this end, it sorts documents of a result list by relevance, producing an ideal DCG at position p ($IDCG_p$), which normalizes the score:

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCGp}.$$

The nDCG values for all queries can be averaged to obtain a measure of the average performance of a ranking algorithm. Note that in a perfect ranking algorithm, the DCG_p will be the same as the $IDCG_p$producing an nDCG of 1.0. All nDCG calculations are then relative values on the interval 0.0 to 1.0 and so are cross-query comparable.

3.4 STEPS IN INFORMATION RETRIEVAL PROCESS

An IR system prepares for retrieval by *indexing documents* (unless the system works directly on the document text) and *formulating queries*, resulting in document representations and query representations, respectively; the system then *matches* the representations and *displays* the documents found and the user *selects* the relevant items. These processes are closely intertwined and dependent on each other. The search process often goes through several

iterations: Knowledge of the features that distinguish relevant from irrelevant documents is used to improve the query or the indexing (*relevance feedback*).

Indexing: Creating Document Representations: Indexing (also called cataloguing, metadata assignment, or metadata extraction) is the manual or automated process of making statements about a document, lesson, person, and so on, in accordance with the conceptual schema (see Figure 4). We focus here on subject indexing – making statements about a document's subjects. Indexing can be *document-oriented* – the indexer captures what the document is about or *request-oriented* – the indexer assesses the document's relevance to subjects and other features of interest to users.

for example, indexing the testimonies in Figure 2 with Jewish-Gentile relations, marking a document as interesting for a course, or marking a photograph as publication quality.

Related to indexing is *abstracting* – creating a shorter text that describes what the full document is about (indicative abstract) or even includes important results (informative abstract, summary). Automatic summarization has attracted much research interest.

Automatic indexing begins with raw feature extraction, such as extracting all the words from a text, followed by refinements, such as eliminating stop words (and, it, of), stemming (pipes Y pipe), counting (using only the most frequent words), and mapping to concepts using a thesaurus (tube and pipe map to the same concept). A program can analyze sentence structures to extract phrases, such as labor camp (a Nazi camp where Jews were forced to work, often for a company; phrases can carry much meaning). For images, extractable features include color distribution or shapes. For music, extractable features include frequency of occurrence of notes or chords, rhythm, and melodies; refinements include transposition to a different key.

Raw or refined features can be used directly for retrieval. Alternatively, they can be processed further: The system can use a classifier that combines the evidence from raw or refined features to assign descriptors from a pre-established index language. To give an example from Figure 2, the classifier uses the words life and model as evidence to assign bioinformatics (a descriptor in Google's directory). A classifier can be built by hand by treating each descriptor as a query description and building a query formulation for it as described in the next section. Or a classifier can be built automatically by using a training set, such as the list of documents for bioinformatics in Figure 2, for machine learning of what features predict what descriptors. Many different words and word combinations can predict the same descriptor, making it easier for users to find all documents on a topic Assigning documents to (mutually exclusive) classes of a classification is also known as text

categorization. Absent a suitable classification, the system can produce one by clustering – grouping documents that are close to each other (that is, documents that share many features).

Query Formulation: Creating Query Representations: Retrieval means using the available evidence to predict the degree to which a document is relevant or useful for a given user need as described in a free-form query description, also called topic description or query statement. The query description is transformed, manually or automatically, into a formal query representation (also called query formulation or query for short) that combines features that predict a document's usefulness. The query expresses the information need in terms of the system's conceptual schema, ready to be matched with document representations. A query can specify text words or phrases the system should look for (free-text search) or any other entity feature, such as descriptors assigned from a controlled vocabulary, an author's organization, or the title of the journal where a document was published. A query can simply give features in an unstructured list (for example, a "bag of words") or combine features using Boolean operators (structured query).

Examples: The Boolean query specifies three ANDed conditions, all of which are necessary (contribute to the document score); each condition can be filled by any of the words joined by OR; one of the words is as good as two or three. If some relevant documents are known, the system can use them as a training set to build a classifier with two classes: relevant and not relevant. Stating the information need and formulating the query often go hand-in-hand. An intermediary conducting a reference interview helps the user think about the information need and find search terms that are good predictors of usefulness. An IR system can show a subject hierarchy for browsing and finding good descriptors, or it can ask the user a series of questions and from the answers construct a query. For buying a digital camera, the system might ask the following three questions:

- What kind of pictures do you take (snapshots, stills, ...)?
- What size prints do you want to make (5x7, 8x10, ...)?
- What computer do you want to transfer images to?

Without help, users may not think of all the features to consider. The system should also suggest synonyms and narrower and broader terms from its thesaurus. Throughout the search process, users further clarify their information needs as they read titles and abstracts.

Matching the query representation with entity representations: The match uses the features specified in the query to predict document relevance. In exact match the system finds the documents that fill all the conditions of a Boolean query (it predicts relevance as 1 or 0).

To enhance recall, the system can use synonym expansion (if the query asks for pipe, it finds tubes as well) and hierarchic expansion or inclusive searching (it finds capillary as well). Since relevance or usefulness is a matter of degree, many IR systems (including most Web search engines) rank the results by a score of expected relevance (ranked retrieval).

Consider the query Housing conditions in Siemens labor camps. Figure 5 illustrates a simple way to compute relevance scores: Each term's contribution is a product of three weights: The query term weight (the importance of the term to the user), the term frequency (tf) (the number of occurrences of the term in the document, synonyms count also), and the rarity of the term or inverse document frequency (idf) on a logarithmic scale. If document frequency = .01 (1 % or 1/100 of all documents include the term), then idf = 100 or 102 and log(idf) = 2. For example, in Figure 5 the contribution of housing to relevance score of Document 1 is query weight 2 * log(idf) 4 * tf (term frequency in document) 5 = 40 (Google considers, in addition, the number of links to a Web page.) Usually (but not in the simple example), scores are normalized to a value between 0 and 1.

Selection: The user examines the results and selects relevant items. Results can be arranged in rank order (examination can stop when enough information is found); in subject groupings, perhaps created by automatic classification or clustering (similar items can be examined side by side); or by date. Displaying title + abstract with search terms highlighted is most useful (title alone is too short, the full text too long). Users may need assistance with making the connection between an item found and the task at hand.

Relevance Feedback and Interactive Retrieval: Once the user has assessed the relevance of a few items found, the query can be improved: The system can assist the user in improving the query by showing a list of features (assigned descriptors; text words and phrases, and so on) found in many relevant items and another list from irrelevant items. Or the system can improve the query automatically by learning which features separate relevant from irrelevant items and thus are good predictors of relevance. A simple version of automatic query adjustment is this: increase the weights of features from relevant items and decrease the weights of features from irrelevant items.

3.5 INFORMATION RETRIEVAL SYSTEM EVALUATION

IR systems are evaluated with a view to improvement (formative evaluation) or with view to selecting the best IR system for a given task (summative evaluation). IR systems can be

evaluated on system characteristics and on retrieval performance. System characteristics include the following:

- The quality of the conceptual schema (Does it include all information needed for search and selection?);
- The quality of the subject access vocabulary (index language and thesaurus) (Does it include the necessary concepts? Is it well structured? Does it include all the synonyms for each concept?);
- The quality of human or automated indexing (Does it cover all aspects for which an entity is relevant at a high level of specificity, while avoiding features that do not belong?);
- The nature of the search algorithm;
- The assistance the system provides for information needs clarification and query formulation; and
- The quality of the display (Does it support selection?).

Measures for retrieval performance (recall, discrimination, precision, novelty) were discussed in the section Relevance and IR system performance. Requirements for recall and precision vary from query to query, and retrieval performance varies widely from search to search, making meaningful evaluation difficult. Standard practice evaluates systems through a number of test searches, computing for each a single measure of goodness that combines recall and precision, and then averaging over all the queries. This does not address a very important system ability: the ability to adapt to the specific recall and precision requirements of each individual query. The biggest problem in IR evaluation is to identify beforehand all relevant documents (the recall base); small test collections have been constructed for this purpose, but there is a question of how well the results apply to large-scale real-life collections. The most important evaluation efforts of this type today are TREC and TDT.

3.6 SUMMARY

In this unit, we have discussed the role of information retrieval systems considering the present day needs. The comparative analysis between data retrieval systems and information retrieval systems is presented. The major objectives of IR systems are discussed in brief. The different types of measures used in evaluation of IR system are presented. The basic steps of IR system are discussed elaborately followed by discussion on IR system characteristics.

3.7 KEYWORDS

Information retrieval, Data retrieval system, Indexing, Querying, Matching, Measures for IR system

3.8 QUESTIONS

1. Discuss the role of IR system for present day applications.
2. Compare IR system to data retrieval systems.
3. What are the objectives of IR system?
4. Discuss the measures used for IR system evaluation.
5. Explain the basic steps in IR process.
6. Discuss the characteristics of IR system.

3.9 REFERENCES FOR FURTHER READING/STUDIES

Singhal, Amit (2001). "Modern Information Retrieval: A Brief Overview". Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35–43.

Korfhage, Robert R. (1997). Information Storage and Retrieval. Wiley. pp. 368 pp.. ISBN 978-0-471-14338-3.

Frakes, William B. (1992). Information Retrieval Data Structures & Algorithms. Prentice-Hall, Inc.. ISBN 0-13-463837-9.

Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich (2008). Introduction to Information Retrieval. Cambridge University Press.

G. Salton, E. Fox, and H. Wu. Extended Boolean Information Retrieval. Communications of the ACM, 1983, 26(11): 1022-1036.

Justin Zobel and Alistair Moffat, Inverted Files for Text Search Engines, ACM Computing Surveys, 38(2), article 6, July 2006.

UNIT 4: SEARCH ISSUES IN INFORMATION RETRIEVAL SYSTEM

Structure

4.0 INTRODUCTION

In general, any Information Storage and Retrieval System is composed of four major functional processes: Item Normalization, Selective Dissemination of Information (i.e., "Mail"), archival Document Database Search, and an Index Database Search along with the Automatic File Build process that supports Index Files. Commercial systems have not integrated these capabilities into a single system but supply them as independent capabilities. Figure 4.1 shows the logical view of these capabilities in a single integrated Information Retrieval System. Boxes are used in the diagram to represent functions while disks represent data storage.

4.1 ITEM NORMALIZATION

Item normalization is the process of canonicalizing items so that matches occur despite superficial differences in the character sequences of the tokens. The most standard way to normalize is to implicitly create equivalence classes, which are normally named after one member of the set. For instance, if the tokens anti-discriminatory and anti-discriminatory are both mapped onto the term anti-discriminatory, in both the document text and queries, then searches for one term will retrieve documents that contain either.

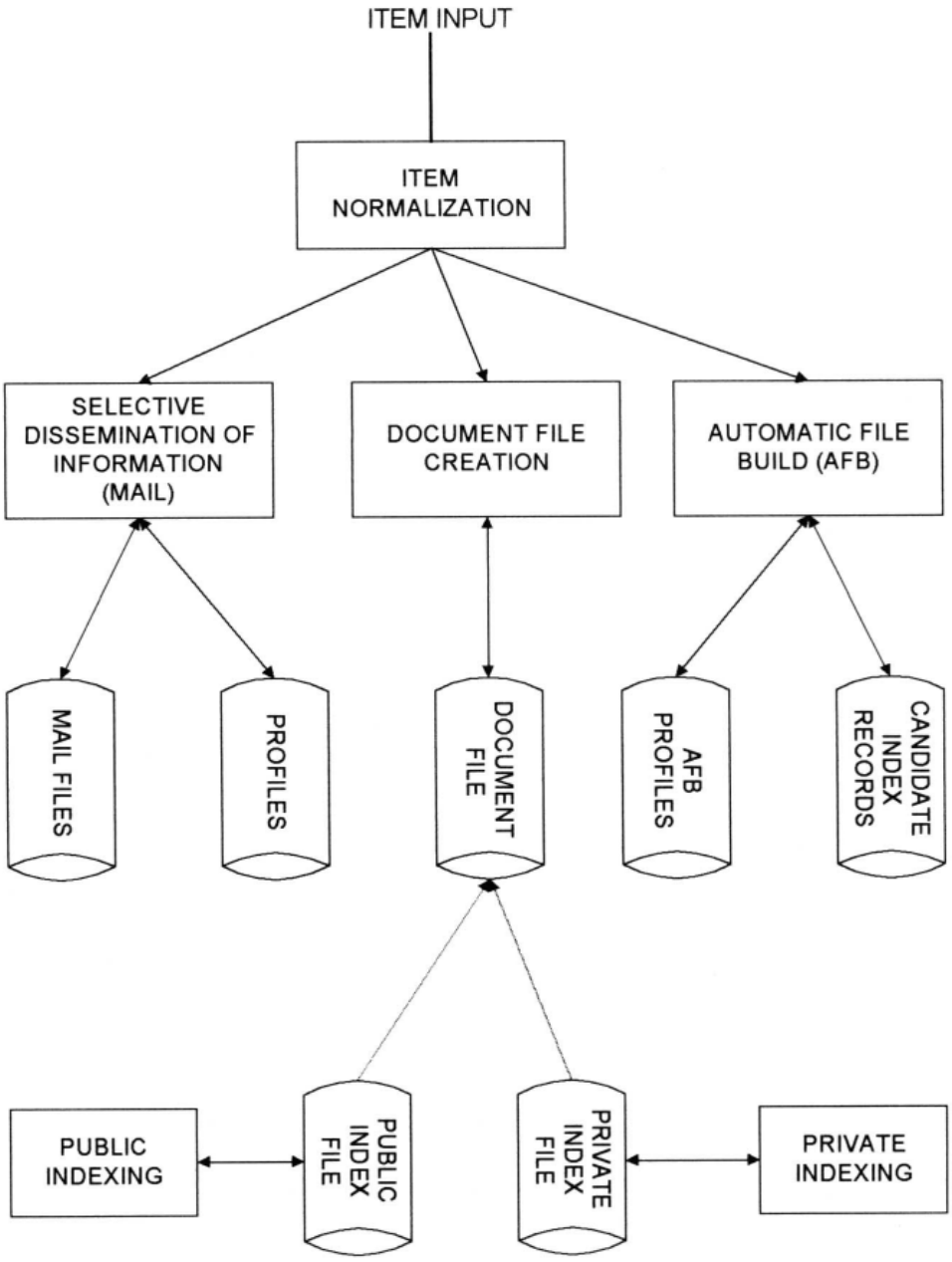


Fig. 4.1. General Structure of Information Storage and Retrieval System (Courtesy: Information Storage and Retrieval Systems- Kowalski and Maybury, 2002)

The advantage of just using mapping rules that remove characters like hyphens is that the equivalence classing to be done is implicit, rather than being fully calculated in advance: the terms that happen to become identical as the result of these rules are the equivalence classes. It is only easy to write rules of this sort that remove characters. Since the equivalence classes are implicit, it is not obvious when you might want to add characters. For instance, it would be hard to know to turn antidiscriminatory into anti-discriminatory. An alternative to creating equivalence classes is to maintain relations between un-normalized tokens. This method can be extended to hand-constructed lists of synonyms such as car and automobile. These term relationships can be achieved in two ways. The usual way is to index un-normalized tokens and to maintain a query expansion list of multiple vocabulary entries to consider for a certain query term. A query term is then effectively a disjunction of several postings lists. The alternative is to perform the expansion during index construction. When the document contains automobile, we index it under car as well (and, usually, also vice-versa). Use of either of these methods is considerably less efficient than equivalence classing, as there are more postings to store and merge. The first method adds a query expansion dictionary and requires more processing at query time, while the second method requires more space for storing postings. Traditionally, expanding the space required for the postings lists was seen as more disadvantageous, but with modern storage costs, the increased flexibility that comes from distinct postings lists is appealing. These approaches are more flexible than equivalence classes because the expansion lists can overlap while not being identical. This means there can be an asymmetry in expansion.

The best amount of equivalence classing or query expansion to do is a fairly open question. But doing a lot can easily have unexpected consequences of broadening queries in unintended ways. For instance, equivalence-classing U.S.A. and USA to the latter by deleting periods from tokens might at first seem very reasonable, given the prevalent pattern of optional use of periods in acronyms. However, if I put in as my query term C.A.T., I might be rather upset if it matches every appearance of the word cat in documents.

Item normalization involves the process of normalizing the incoming items to a standard format. In addition to translating multiple external formats that might be received into a single consistent data structure that can be manipulated by the functional processes, item normalization provides logical restructuring of the item. Additional operations during item normalization are needed to create a searchable data structure: identification of processing

tokens (e.g., words), characterization of the tokens, and stemming (e.g., removing word endings) of the tokens. The original item or any of its logical subdivisions is available for the user to display. The processing tokens and their characterization are used to define the searchable text from the total received text. Figure 4.2 shows the normalization process.

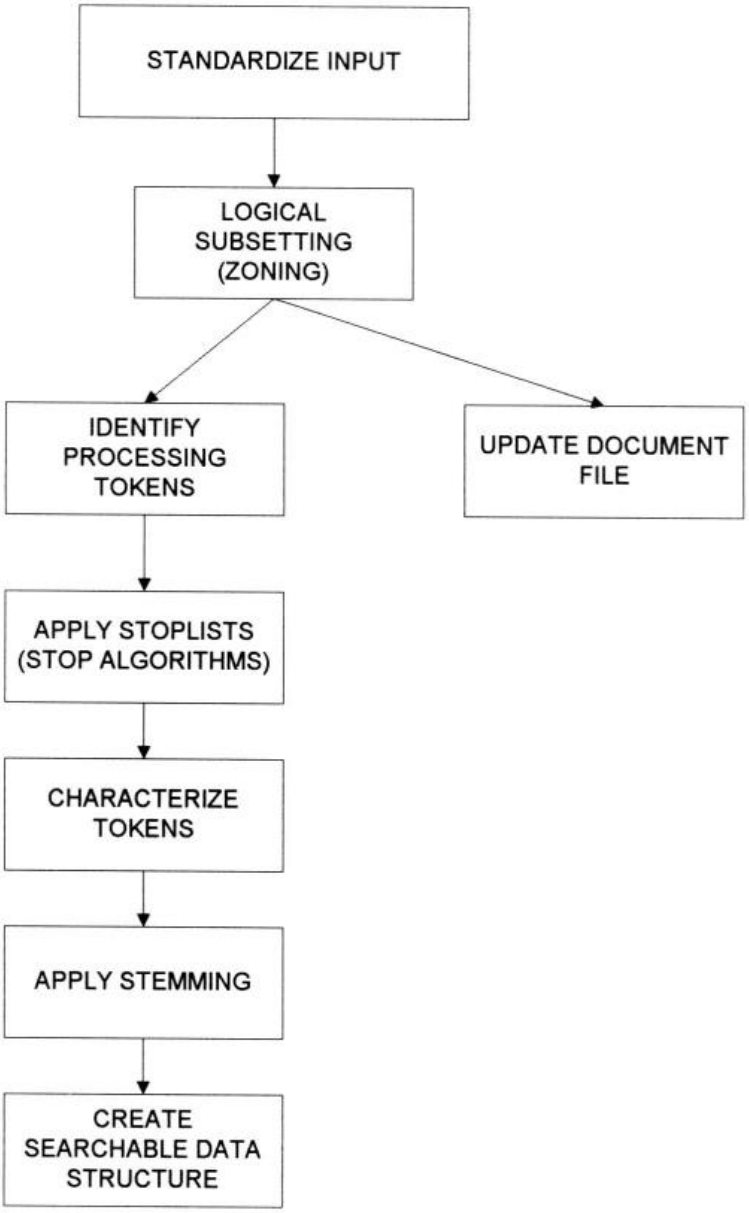


Fig. 4.2. Text Normalization Process

Standardizing the input takes the different external formats of input data and performs the translation to the formats acceptable to the system. A system may have a single format for all items or allow multiple formats. One example of standardization could be translation of foreign languages into Unicode. Every language has a different internal binary encoding for

the characters in the language. One standard encoding that covers English, French, Spanish, etc. is ISO-Latin. There are other internal encodings for other language groups such as Russian (e.g, KOI-7, KOI-8), Japanese, Arabic, etc. Unicode is an evolving international standard based upon 16 bits (two bytes) that will be able to represent all languages. Unicode based upon UTF-8, using multiple 8-bit bytes, is becoming the practical Unicode standard. Having all of the languages encoded into a single format allows for a single browser to display the languages and potentially a single search system to search them.

Multi-media adds an extra dimension to the normalization process. In addition to normalizing the textual input, the multi-media input also needs to be standardized. There are a lot of options to the standards being applied to the normalization. If the input is video the likely digital standards will be either MPEG-2, MPEG-1, AVI or Real Media. MPEG (Motion Picture Expert Group) standards are the most universal standards for higher quality video where Real Media is the most common standard for lower quality video being used on the Internet. Audio standards are typically WAV or Real Media (Real Audio). Images vary from JPEG to BMP. In all of the cases for multi-media, the input source is encoded into a digital format. To index the modal different encodings of the same input may be required. But the importance of using an encoding standard for the source that allows easy access by browsers is greater for multi-media then text that already is handled by all interfaces. The next process is to parse the item into logical sub-divisions that have meaning to the user. This process, called "Zoning," is visible to the user and used to increase the precision of a search and optimize the display. A typical item is sub-divided into zones, which may overlap and can be hierarchical, such as Title, Author, Abstract, Main Text, Conclusion, and References. The term "Zone" was selected over field because of the variable length nature of the data identified and because it is a logical sub-division of the total item, whereas the term "fields" has a connotation of independence. There may be other source-specific zones such as "Country" and "Keyword." The zoning information is passed to the processing token identification operation to store the information, allowing searches to be restricted to a specific zone. For example, if the user is interested in articles discussing "Einstein" then the search should not include the Bibliography, which could include references to articles written by "Einstein." Zoning differs for multi-media based upon the source structure. For a news broadcast, zones may be defined as each news story in the input. For speeches or other programs, there could be different semantic boundaries that make sense from the user's perspective.

Once a search is complete, the user wants to efficiently review the results to locate the needed information. A major limitation to the user is the size of the display screen which constrains the number of items that are visible for review. To optimize the number of items reviewed per display screen, the user wants to display the minimum data required from each item to allow determination of the possible relevance of that item. Quite often the user will only display zones such as the Title or Title and Abstract. This allows multiple items to be displayed per screen. The user can expand those items of potential interest to see the complete text.

Once the standardization and zoning has been completed, information (i.e., words) that are used in the search process need to be identified in the item. The term processing token is used because a "word" is not the most efficient unit on which to base search structures. The first step in identification of a processing token consists of determining a word. Systems determine words by dividing input symbols into three classes: valid word symbols, inter-word symbols, and special processing symbols. A word is defined as a contiguous set of word symbols bounded by inter-word symbols. In many systems inter-word symbols are non-searchable and should be carefully selected. Examples of word symbols are alphabetic characters and numbers. Examples of possible inter-word symbols are blanks, periods and semicolons. The exact definition of an inter-word symbol is dependent upon the aspects of the language domain of the items to be processed by the system. For example, an apostrophe may be of little importance if only used for the possessive case in English, but might be critical to represent foreign names in the database. Based upon the required accuracy of searches and language characteristics, a trade off is made on the selection of inter-word symbols. Finally there are some symbols that may require special processing. A hyphen can be used many ways, often left to the taste and judgment of the writer (Bernstein-84). At the end of a line it is used to indicate the continuation of a word. In other places it links independent words to avoid absurdity, such as in the case of "small business men." To avoid interpreting this as short males that run businesses, it would properly be hyphenated "small-business men." Thus when a hyphen (or other special symbol) is detected a set of rules are executed to determine what action is to be taken generating one or more processing tokens.

Next, a Stop List/Algorithm is applied to the list of potential processing tokens. The objective of the Stop function is to save system resources by eliminating from the set of searchable processing tokens those that have little value to the system. Given the significant increase in

available cheap memory, storage and processing power, the need to apply the Stop function to processing tokens is decreasing. Nevertheless, Stop Lists are commonly found in most systems and consist of words (processing tokens) whose frequency and/or semantic use make them of no value as a searchable token. For example, any word found in almost every item would have no discrimination value during a search. Parts of speech, such as articles (e.g., “the”), have no search value and are not a useful part of a user’s query. By eliminating these frequently occurring words the system saves the processing and storage resources required to incorporate them as part of the searchable data structure. Stop Algorithms go after the other class of words, those found very infrequently.

4.2 DOCUMENT DATABASES

Increasingly unstructured or semi-structured documents are the drivers of new, novel decision support systems. In the expanded framework, DSS linked to a document database are called document-driven DSS. A number of approaches can be used to store and retrieve documents for decision support including: 1) storing the documents in directories or files, 2) using a RDBMS to store documents or some document metadata with a link to the complete document, 3) using a document-oriented database. Document databases are a type of "NoSQL" or XML database. Documents may be stored using markup, XML, PDF and Microsoft Office formats.

Documents are the organizing structure in a document database. Conceptually a document is similar to records or rows in relational databases, but they are less structured. Documents do not adhere to a standard schema with structured fields, sections, slots, parts, or keys. According to Krishnan, "Document-based databases do not store data in tables with uniform sized fields for each record. Instead, each record is stored as a document that has certain characteristics. Any number of fields of any length can be added to a document."

Wikipedia has the following example of a document:

FirstName="Bob", Address="5 Oak St.", Hobby="sailing"
Another document could be:
FirstName="Jonathan", Address="15 Wanamassa Point Road", Children= [{Name: "Michael", Age:10}, {Name: "Jennifer", Age:8}, {Name: "Samantha", Age:5}, {Name: "Elena", Age:2}]

"Both documents have some similar information and some different. Unlike a relational database where each record would have the same set of fields and unused fields might be kept empty, there are no empty 'fields' in either document (record) in this case. This system allows new information to be added and it doesn't require explicitly stating if other pieces of information are left out."

Cattell (2011) identifies 3 types of NoSQL data stores -- key-value store, document store and extensible record store.

According to Cattell, "Key-value Store provide a distributed index for object storage, where the objects are typically not even interpreted by the system: they are stored and handed back to the application as BLOBs."

"Document Stores provide more functionality," according to Cattell. For example, "the system does recognize the structure of the objects stored. Objects (or documents) may have a variable number of named attributes of various types (integers, strings), objects can grouped into collections, and the system provides a simple query mechanism to search collections for objects with particular attribute values."

Finally, "extensible Record Stores, sometimes called wide column stores, provide a data model more like relational tables, but with a dynamic number of attributes, and like document stores, higher scalability and availability made possible by database partitioning and by abandoning database-wide ACID semantics."

According to Ayende Rahien, "A document database is, at its core, a key/value store with one major exception. Instead of just storing any blob in it, a document db requires that the data will be store in a format that the database can understand. The format can be XML, JSON, Binary JSON (MongoDB), or just about anything, as long as the database can understand it."

Ayende notes "A document database is schema free, that is, you don’t have to define your schema ahead of time and adhere to that. It also allows us to store arbitrarily complex data.

A major limitation of document databases is limited query capabilities. Ho (2009) notes "Many of the NoSQL DB today are based on the DHT (Distributed Hash Table) model, which provides hash table access semantics. To access or modify any object data, the client is required to supply the primary key of the object, then the DB will look up the object using an equality match to the supplied key." Developers need to organize the indexing.

In addition, most thinking about databases tends to relate to the use of tables and transactional relationships, and generally results in creating transaction oriented Relational Database Management Systems.

There is also a lot of data that would be structured quite differently, namely in the form of linked documents:

- Usenet News articles
- Sets of documents on my computer
- Research papers at a university
- All addressable web pages on the Internet
- The set of engineering drawings used to help repair American Airline's aircraft

These "pieces of data" that would typically be called documents certainly have structure, but not of a sort that can be sufficiently rigidly defined as to be conveniently represented as a set of relational tables.

In the case of `legacy' documents (e.g. - documents not designed with re-accessability in mind), there may indeed be little or no structure that can be recognized and used in an automated fashion.

Many organizations don't recognize that their overall set of documents in fact represents a database that is valuable and worth managing. They only find this out when something horrible happens such as when a LAN "goes down" and destroys a large number of critical documents.

A wide variety of tools are available for structuring, managing and searching these sorts of "document databases," both in commercial and free realms.

4.3 Search Tools

For the most simple of searching through text, one might use a program such as Unix's grep command. It's not terribly sophisticated, but it and such variants as agrep (for doing "approximate" pattern matching) or sgrep (for searching SGML documents) are nonetheless tremendously useful for searching for things, particularly when composed with other commands.

Many "search engines" have been designed to index hierarchies of text database material, and provide far more sophisticated tools to query information.

Remembrance Agent source code in C and Emacs LISP is available; RA integrates with the Emacs editor to automagically retrieve data that is "related" to whatever you're working on now.

Beagle: A Gnome tool written using Mono ; see also Beagle search tool. The KDE tools for Beagle have appeared; Kerry - KDE frontend for Beagle and yaBi - beagle search client for KDE.

4.4 MULTIMEDIA DATABASES

A multimedia database is a database that hosts one or more primary media file types such as .txt (documents), .jpg (images), .swf (videos), .mp3 (audio), etc. And loosely fall into two main categories:

- Static media (time-independent, i.e. images and handwriting)
- Dynamic media (time-dependent, i.e. video and sound bytes)

4.4.1 Data types

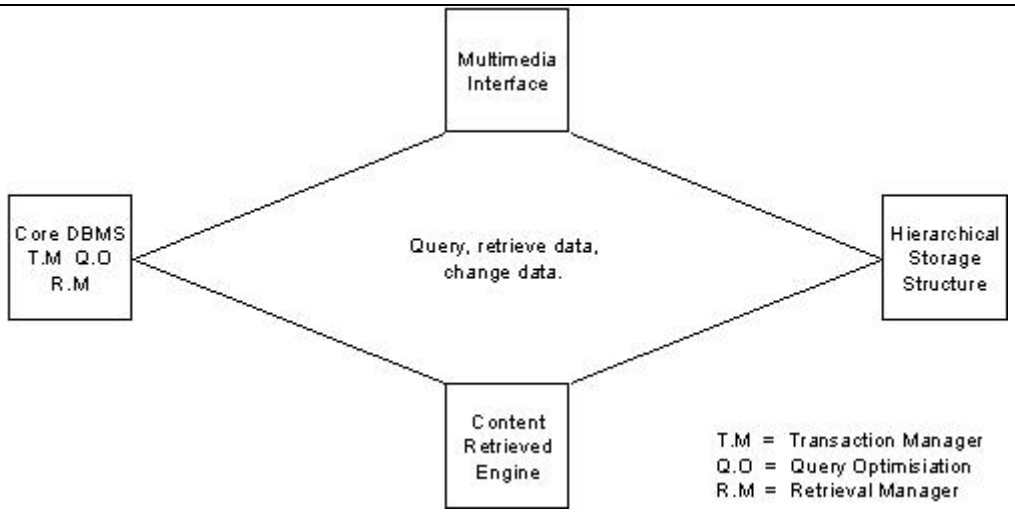
In addition to the standard numeric, date and text data types, there are a number of data types that are regarded as the basic building blocks of MM applications. These data types, which are elements of more complex MM objects, are:

- Text - different fonts and to produce special effects such as colour and fill.
- Audio - various audio file formats include Microsoft WAV (wave) and MIDI, which is a more compact representation of sound.
- Still images - pixels can be 0 or 1 ('white' or 'black') or hi-res colour images with 8, 16 or 24 bits per pixel.
- Digital video - usually stored as a sequence of frames. For realistic playback, the transmission, compression, and decompression of digitized continuous frames requires transfer rates of 30 frames per second. If audio is required as well, the audio and video must be interleaved so that the sequences can be timed properly. Microsoft's AVI format can synchronize playback of audio and video.
- Graphical objects - such as 2- and 3-dimensional images.

Most of these data types require a lot of storage space. An average page of text may require about 2 KB; 75 minutes of high-fidelity music may need 100MB; a full page, still image varies from about 10KB for black-and-white to several megabytes for colour; a video frame may require 1 MB of storage so a video clip lasting a second needs something of the order of 3 0MB. Compression techniques are available to reduce these storage requirements but it will be clear that MM still needs a lot of storage space.

MMDBs require all the basic attributes of a database management system such as a transaction manager, query optimizer, recovery manager etc. as well as special storage structures and specialized search and querying modules.

4.4.2 Architecture of a MMDB application



Since existing relational and OO databases comprise the basic requirements of any database, it is natural that many multimedia and imaging DB applications are constructed within such existing systems. In order to support such applications, many DBMS vendors offer facilities suitable for MM. These include:

- long bit and byte strings
- BLOBS
- Paths or references of images where the actual image stored elsewhere, such as on an optical storage subsystem. The reasons for this are that document imaging systems need on-line, near-line and off-line storage of images, including archiving. This may be achieved by the use of optical jukeboxes but most commercial DBMSs do not directly support optical storage subsystems (Informix Online/optical is an exception).
- Content retrieval capabilities. In conventional relational and OO DBs querying is based on the attributes of objects. However, information retrieval and document imaging systems require searching the content of documents. This ability can be generalized to still images, audio and video.

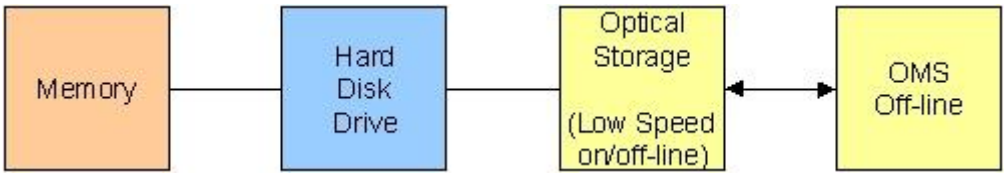
4.4.3 Characteristics of Multimedia and Imaging Databases

1. Support for imaging and multimedia data types plus
2. the capacity to handle many MM objects plus
3. support for suitable storage management plus
4. database capabilities (transaction/concurrency control, integrity, etc) plus
5. information retrieval capabilities (including exact match and probabilistic retrieval)

4.4.4 Hierarchical Storage Management

It has already been pointed out that MM objects can require a lot of memory for storage. It will also be necessary to have an appropriate storage mechanism so that the system can keep track of objects that are swapped between near-line and on-line and inform the user when an object is stored off-line. In order to do this, the preferred mechanism is that of hierarchical storage management. This is based on the idea of managing a hierarchy of on-line, near-line and off-line storage media. Each of these levels has a particular performance. capacity and cost.

- RAM - best performance, smallest capacity, highest cost, little permanence.
- Hard-drive - good performance, reasonable capacity. fairly high cost, some online storage capabilities.
- Optical storage - on-line with a drive or near-line with a jukebox. Acceptable performance when on-line but slow when near-line. High capacity, reasonable cost (less than preceding levels). Can be used for archiving which is permanent e.g. WORMdevices, CD-ROM and recordable compact discs.
- Optical media stored off line - stored in cabinets, on shelves etc. Unlimited capacity. very cheap, lasts much longer than magnetic media and therefore good for archiving. Poor performance in the sense that the user has to take the discs off the shelf and put them in the drive!



4.4.5 Requirements for a MMDBMS

A prime requirement for a multimedia DBMS is to manage this hierarchy of devices. Through caching and archival capabilities, objects that are no longer accessed are automatically stored on slower media in the hierarchy and eventually may be taken off-line. It is possible for advanced relational, object-relational and object-oriented DBMSs to cope with these requirements to some extent by assigning large segments or clusters to store multimedia objects. The segments are then mapped onto disc volumes and the application developer can use volumes on the storage subsystem to manage the multimedia object. However, there are problems with very large databases. Treating a volume as a large storage space without considering the performance and functionality of the storage subsystem will cause serious problems or result in the system not working at all.

- **Searching for and retrieving data:** Although there are similarities with retrieving data from conventional DBs, in MMDBs it is often necessary to find objects that satisfy the user's query as closely as possible rather than finding an exact match. It should be possible to rank the results of searches. Queries may involve record-attribute searching as well as content-based searching and the query optimizer must take this into account.

- **Spatial data types:** In many MM systems elements may have a spatial relationship with one another e.g. in GISs. The user may wish to query the DB using spatial predicates such as location, position with respect to others (object 1 is to left of object2 or object 1 is contained within object2, etc.). Such MMDBMSs require support for queries of this type.

- **Interactive querying:** How do you query a MM object? If you know that you want the whole of a particular named image or a particular video clip then the query is straightforward. However, frequently the query may be more complex and require some interactive exchange between the user and the DB as the user attempts to refine his query. In MMDBs it is common to have domains or pick lists of various existing objects so that the user can construct a query interactively. Query-by-example functions may also be available where the user builds an example MM object from existing domain elements. For example, the user may first ask to be shown images of all the types of

fabrics stored using a GUI. They might then pick a fabric and say: show me all the fabrics of similar texture and different colours.

- Content retrieval is not as precise as querying records or object attributes and various weights may be applied to extracted features. For example, values of important features such as colour, texture or shape are weighted and the distance between these sets of attributes in the sample and returned images is computed. This can be used as a measure of how closely the returned images fit the query example image. Another approach is demonstrated by Excalibur, a system which recognises patterns using a neural network. For example, the user may provide an image, such as a fingerprint, and ask the system to find those images that match it exactly or are nearest to it.
- **Automatic feature extraction and indexing:** When records are inserted into a relational database the attribute values of the object must be specified precisely. Indexes are normally specified by the user or DB designer. However, for MM systems, tools may be available to extract the important features of MM objects and even automatically produce indexes. For example, with advanced document management systems, paper documents may be converted into scanned digitized images that are subsequently recognized by OCR products and the contents may be automatically indexed. Hence, the bulk of the attribute, or feature, data entry is performed automatically.

4.4.6 Performance issues

MMDBMSs must be able to provide good performance for real-time querying and updating. Some of the features that influence this are:

- Indexing - most DBs use single key access structures such as B-trees which can be used for retrieving ranges as well as precise matches. For MM purposes, spatial and multidimensional indexes are also useful. Two dimensional objects have X and Y co-ordinates (multi-dimensional objects will have more) and special structures such as R trees or grid files will provide better access time.
- Content-retrieval indexing - special indexes are required for this. For example, the index for a video could contain the frame number of the start of each clip or scene.
- Organising BLOBs - interfaces for BLOBs may allow the user to access and update byte or bit streams and so positional indexes are required to provide fast access to continuous streams of bytes or bits starting at a certain position.

- Query optimization - MMDBs is large and manages many complex objects. Query optimization is vital for providing reasonable performance.

4.4.7 Relational Databases and Multi-media

It should be clear by now that, although MMDBMSs require many of the features that are also possessed by relational database management systems, they also have many requirements that are not met by conventional systems. However, in view of the current dominance of relational systems, it is reasonable to consider whether relational DBMSs can be extended to provide at least some MM capabilities.

Variable length fields: Most of the data types supported by relational DBs are fixed length and so the length of each record is also fixed However, many database vendors also provide variable length fields with the aim of supporting at least some MM data types - text, digitized audio, still images etc. (You are familiar with Oracle's VARCHAR2, for example, and various vendors provide data types such as BLOBs, IMAGE. CHARACTER VARYING). Unfortunately, there is no uniformity about these. Even with varying length character fields there are various maximum sizes - some may be as little as 32 or 64 KB per field value which is not much use for most MM purposes (see figures quoted earlier for storage requirements). However, such small fields can be useful for memos or other simple text additions to records.

SQL92 provides some support for variable length attributes and also provides BIT and BIT VARYING for storing bit-mapped graphics. SQL3 will provide more support for MM data types when it finally appears.

BLOBS: Classical DB theory says that fields in a relational database must be atomic. However, for large MM data it is not reasonable to assume that all the bytes in the long field will be read or updated as a whole. Although many vendors provide support for BLOBS through various data types (BLOB, VARCHAR VARYING. BIT VARYING) there is no single convention for manipulating such fields.

Embedding SQL statements in application programming languages(e.g. C or C-H-). In these applications, data is interchanged between SQL and variables set up in the host programming language. There are several ways of doing this - for example, rows of data may be processed one at a time using a cursor operation or data may be inserted, updated, deleted without using

a cursor. An application programming interface (API) is required that controls the exchange of information with the database server that manages the SQL database.

One of the key features of APIs is that they should be able to access BLOBs from a database and integrate them with front-end applications that can manipulate large quantities of space-intensive MM data. Clearly, it would be useful if substrings of the MM data could be retrieved and updated - apart from anything else, this would reduce the overhead (buffer or cache space) and communication.

As mentioned above, various mechanisms are used for retrieving values from a database to a host programming language and there is no need to go into details here. The main issues involved in manipulating these variable-length fields are:

- size is not known in advance
- the data stored in these field may be quite large and cannot be read as a whole. So the data must be managed a piece at a time and byte strings read as needed.

4.4.8 Examples of MM systems based on RDBMS

InterBase

This relational database system has built in support for BLOBs. The BLOBs are stored in collections of 'segments' - the size of which can be specified by the user and are basically a fixed length 'page' or I/O block.. InterBase has a proprietary high-level language programme interface as well as a standard SQL interface. The individual segments that comprise a BLOB can be read and or updated.

Sybase SQL server allows users to declare columns as TEXT and IMAGE data types which can be very large (2GB). Sybase has an enhanced version of SQL called TransactSQL which allows some manipulation of the TEXT and IMAGE data types such as finding the first occurrence of a particular "pattern" in the column. The column values of TEXT/IMAGE contain pointers to the first page of the MM column and these pages are stored separately from the tables for the database. The pages on which the object is stored form a linked list.

XDP from Plexus:

This is an imaging database engine based on the INFORMIX Turbo relational DBMS. Unlike Sybase, this system does support hierarchical storage subsystems and manages magnetic discs, optical discs, and optical jukeboxes with on-line, near-line and off-line facilities. Records and the images associated with them are stored in different locations but both image/text structures and records can be manipulated and updated consistently in the same transaction.

4.5 DIGITAL LIBRARIES AND DATA WAREHOUSING

With the dissemination of the Internet, a great amount of documents is available for search and retrieval on the Web. The Internet is now one of the biggest information repositories. However, its content is disorganized and distributed. Moreover, the diverse hardware and software platforms, as well the different document formats and diverse media available compose a great heterogeneous database, which contains structured, half-structured and non-structured data. All this distribution and heterogeneity have contributed to make the search and the Web content acquisition difficult. In this context, Digital Libraries (DL), a recent research area, aims at organizing and promoting an easier access to documents on the Web.

As there are many definitions in the literature and there is no consensus regarding the DL concept. A DL is considered to be a great object collection, in diverse digital formats, persistent, managed and well organized using a catalogue and with access through the Web. The development of a DL generally implies in integration of distributed multimedia content on the Web. Since the hypermedia nature of the Web implies in navigation through the content in order to get the desired information, the organization of the integrated data should consider a content categorization into hierarchies.

There are various initiatives which aim at developing DL whose proposal is to solve the problem of content integration as well as the access to these contents using hierarchical classification. The following projects are some related works: Digital Stanford Library Technologies [PBCCG2000], Digital Illinois Library Initiative Project [Chen2000], Digital Alexandria Library Project [ACDFF+1995] and University of Digital Michigan Library Project [WB1998]. However, it is noted that such initiatives do not present a comprehensive proposal to address the issues related to DL.

A research area that has been contributing to solve complex database problems is the area of Data Warehousing (DWing). The DWing approach has been very useful to address issues related to data integration and complex search.

The process of digital library development includes issues such as the integration of complex documents found on the Web. Moreover, access to the DL must be assisted by the use of content hierarchies that guide the user in the discovery and filtering of information of his/her interest. In some research work more emphasis is being given to the item of integration of complex and heterogeneous data, using approaches such as CORBA, agents and mediators.

Some works emphasize the need for a systematic approach that allows the automation of the main typical library functions, such as classification, cataloguing, etc.

4.6 DATA WAREHOUSING APPROACH IN THE DIGITAL LIBRARY DEVELOPMENT

The DL development based on the Dwing approach implies in understanding the DWing architecture and how to use and/or adapt its processes and components for the DL.

Data Warehousing

In accordance with William H. Inmon [Inmon1996], a DW has the following characteristics:

- It is subject-oriented (the data are stored in accordance with specific areas of the business or specific subjects/aspects of the company interest).
- It is integrated (i.e., it integrates data from diverse sources, while identifying and correcting inconsistencies).
- It is a collection of non-volatile data (it means that data are loaded and accessed, but its updating does not occur in a DW environment).
- It is variant in the time (the time horizon for DW is significantly longer than that of production systems; the data consist of a sophisticated series of "snapshots" obtained at a certain moment; and the DW key structure always consists of some temporal elements).
- It is used for supporting management decisions.

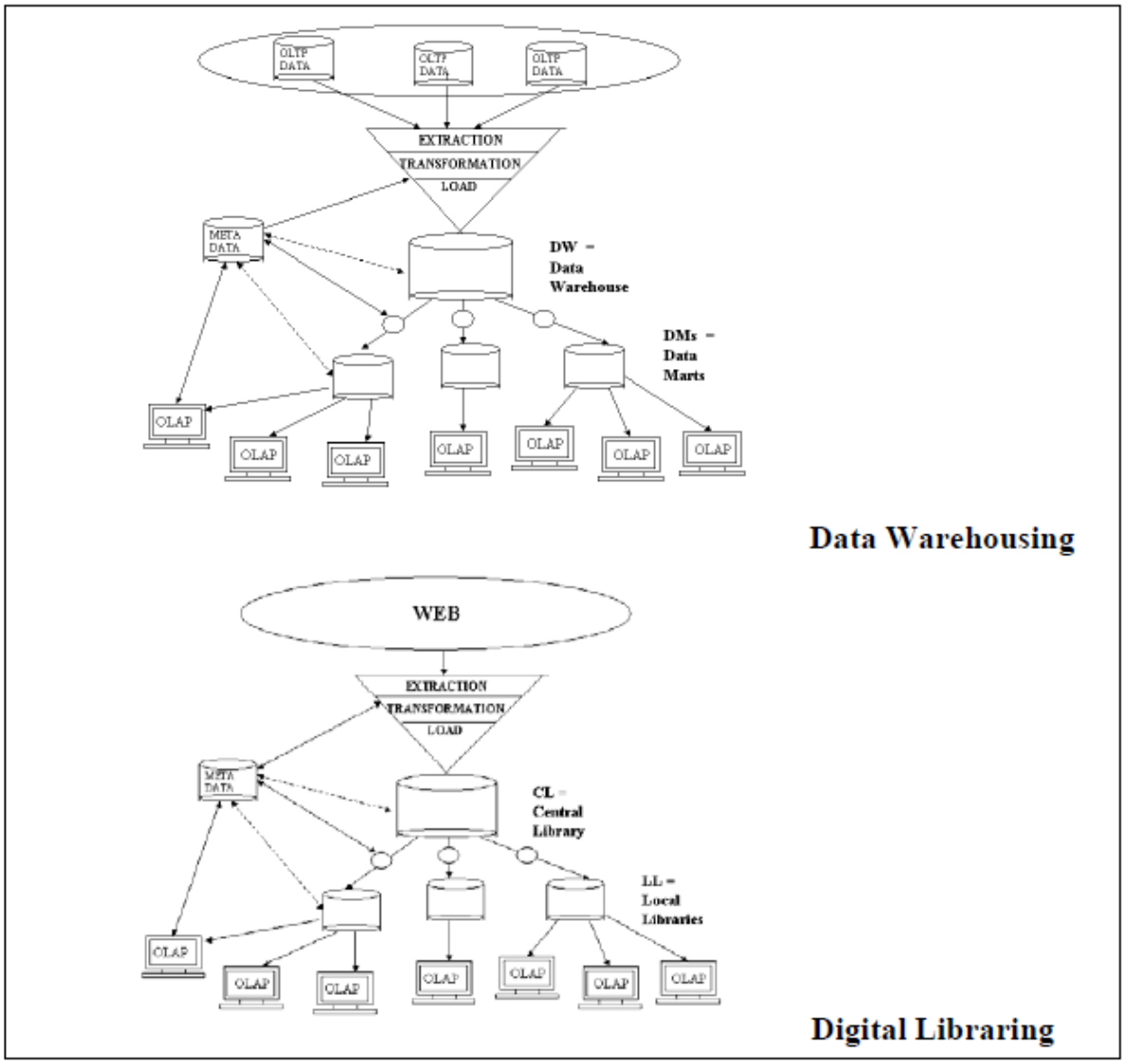
Generally, architecture for systems based on DW involves the integration of current and historical data. The data sources can be internal (operational systems of the company/institution) or external (containing complementary data originated from the organization, such as economic indicators). Generally, the data integration deals with different data models, definitions and/or platforms. This heterogeneity demands the existence of applications that extract and transform data in a way that the data integration becomes possible. Once integrated, the new data are stored in a new database - DW - that combines different points of view for supporting management decisions. This database is used for data analysis by final users. DW can be divided into some databases called Data Marts (DM). Such DMs contain information that can be useful to different departments of the company. They are also considered as departmental DWs. DM/DW can be accessed by OLAP (Online Analytical Processing) or DMining tools and/or DSS (Decision Support Systems). These tools make the data navigation possible, as well as the managerial analysis and the knowledge discovery. An important component of this architecture is the metadata repository, where the information about the DW development can be found.

4.7 DIGITAL LIBRARY FROM THE DATA WAREHOUSING APPROACH

By applying the DW-related concepts shown above in the DL process, we observe that:

- a DL must be subject-oriented (as mentioned previously, it is important for the users that they can search for documents through a subject hierarchical classification).
- a DL must have an integrated view of documents. A possible distribution of these documents, as well as inconsistencies, must be transparent to the final user.
- Documents and its corresponding metadata must be loaded only one time in the DL and its contents do not have to be updated; the users access are for reading only
- The documents are stored in the DL and other versions can be enhanced. Moreover, the documents generally have a temporal orientation related to the publication date. So, the temporal aspect is also of interest in a DL.
- Finally, although a DL is not necessarily used to support management decisions, it is used to support the process of decision-making in the research. Thus, the decision-support characteristic is also of interest.

Another aspect that is important to observe is the distinction between central and local libraries, which becomes possible in the proposed approach through the differentiation between DW and DM. DW refers to the Central Library while DM refers to the Local (Departmental) Libraries.



As shown in Figure 4.1, in the process of the DL development (DLing), the process of a DW development (DWing) occurs as follows:

a. The data sources are not previously defined and do not consist of transactional data sources of a given company (which are generally legacy systems and relational databases). They are composed by available documents on the public or private Web instead.
b. The extraction process, instead of using conventional data extraction tools from databases or legacy archives, is made through the process of document search on the Web and its filtering. The information are accessed through traditional searching mechanisms such as Yahoo, Altavista, Google etc., or even by the mechanisms created specially to this end which can look for documents on the hidden Web

[RG2001]. An additional stage consists of the filtering of documents that are of real interest to the user.

c. The transformation process consists of an analysis of the documents obtained in the previous process, capturing their metadata that are necessary to the load process of the database and will compose the library catalogue.
d. The load process, beyond effectively generating the referring catalogue of the captured documents, makes a copy of the documents found.
e. The DW contains documents of all the subjects of interest to a given institution, as well as their respective metadata which are organized in hierarchies according to the ontology.
f. The DM contains the documents metadata (catalogues) of all subjects relating to a department for which such database was generated.
g. The search for documents and the catalogue (DW) visualization use OLAP navigation techniques making it possible to find those whose characteristics are of interest to the user.

4.8 SUMMARY

In this unit, we have described the role of information system for present day needs and its importance. The process of item normalization and the concept of document databases are explained in detail. The role of multimedia databases along with its data types, characteristics, architecture are discussed in detail. The role of digital library and its development through data warehousing is addressed and its architecture is given from design perception.

4.9 KEYWORDS

Information retrieval, Item normalization, Document databases, Multimedia databases, Digital library, Data warehousing

4.10 QUESTIONS

1. With the general structure of IR system, explain item normalization process.
2. Discuss the procedure of text normalization process
3. What are document databases? Name the search tools used in document databases.
4. Define multimedia databases? Write a note on MM data types.
5. With the architectural block diagram of MMDB, explain each component in brief.

6. What are the characteristics of MM databases?
7. Explain the hierarchical storage management system.
8. What are the requirements for a MMDBMS?
9. Explain the performance issues to be addressed in MMDBMS.
10. Discuss the role of digital libraries in brief.
11. Describe the development of digital library from the data warehousing approach.

4.11 REFERENCES FOR FURTHER READING/STUDIES

Andresen, D., Carver, L., Dolin, R., Fischer, C., Frew, J., Goodchild, M., Ibarra, O., Kothuri, R., Larsgaard, M., Manjunath, B. S., Nebert, D., Simpson, J., Smith, T. R., Yang, T., Zheng, Q.; "The WWW prototype of the Alexandria Digital Library", In the Proceedings of ISDL'95: International Symposium on Digital Libraries, Japan, 1995.

Chen, H.; "The Illinois Digital Library Initiative Project: Federating Repositories and Semantic Research", 2000.

Frakes, William B. (1992). Information Retrieval Data Structures & Algorithms. Prentice-Hall, Inc.. ISBN 0-13-463837-9.

G. Salton, E. Fox, and H. Wu. Extended Boolean Information Retrieval. Communications of the ACM, 1983, 26(11): 1022-1036.

Inmon, W.H.; Building the Data Warehouse, John Wiley & Sons, Inc., 1996

Justin Zobel and Alistair Moffat, Inverted Files for Text Search Engines, ACM Computing Surveys, 38(2), article 6, July 2006.

Korfhage, Robert R. (1997). Information Storage and Retrieval. Wiley. pp. 368 pp.. ISBN 978-0-471-14338-3.

Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich (2008). Introduction to Information Retrieval. Cambridge University Press.

Paepcke, A., Baldonado, M., Chang, C.K., Cousins, S., Garcia-Molina, H.; "Building the InfoBus: A Review of Technical Choices in the Stanford Digital Library Project", 2000.

Singhal, Amit (2001). "Modern Information Retrieval: A Brief Overview". Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35–43.

Weinstein, P., Birmingham., W.; "Organizing Digital Library Content and Services with Ontologies", International Journal on Digital Libraries, special issue on artificial intelligence for digital libraries,1998.

UNIT 5: INFORMATION RETRIEVAL SYSTEM CAPABILITIES

Structure

5.0 INTRODUCTION

The major functions that are available in an Information Retrieval System are discussed in this section. Search and browse capabilities are crucial to assist the user in locating relevant items. The search capabilities address both Boolean and Natural Language queries. The algorithms used for searching are called Boolean, natural language processing and probabilistic. Probabilistic algorithms use frequency of occurrence of processing tokens

(words) in determining similarities between queries and items and also in predictors on the potential relevance of the found item to the searcher. Given the imprecise nature of the search algorithms, Browse functions to assist the user in filtering the search results to find relevant information are very important.

5.1 Search Capabilities

The objective of the search capability is to allow for a mapping between a user's specified need and the items in the information database that will answer that need. The search query statement is the means that the user employs to communicate a description of the needed information to the system. It can consist of natural language text in composition style and/or query terms with Boolean logic indicators between them.

Given the following natural language query statement where the importance of a particular search term is indicated by a value in parenthesis between 0.0 and 1.0 with 1.0 being the most important:

Find articles that discuss automobile emissions(.9) or sulfur dioxide(.3) on the farming industry.

The system would recognize in its importance ranking and item selection process that automobile emissions are far more important than items discussing sulfur dioxide problems. Many different functions are associated with the system's understanding the search statement based on the different algorithms. The functions define the relationships between the terms in the search statement (e.g., Boolean, Natural Language, Proximity, Contiguous Word Phrases, and Fuzzy Searches) and the interpretation of a particular word.

5.1.1 Boolean Logic

The principle of Boolean logic lets you organize concepts together to define what information is needed. When searching the databases, these sets are controlled by use of Boolean operators OR, AND, and NOT. These operations are implemented using set intersection, set union and set difference procedures. Placing portions of the search statement in parentheses

are used to specify the order of Boolean operations (i.e., nesting function). If parentheses are not used, the system follows a default precedence ordering of operations (e.g., typically NOT then AND then OR). A special type of Boolean search is called "M of N" logic. The user lists a set of possible search terms and identifies, as acceptable, any item that contains a subset of the terms. For example, "Find any item containing any two of the following terms: "AA," "BB," "CC." This can be expanded into a Boolean search that performs an AND between all combinations of two terms and "OR"s the results together ((AA AND BB) or (AA AND CC) or (BB AND CC)).

5.1.2 Proximity

Proximity is used to restrict the distance allowed within an item between two search terms. The semantic concept is that the closer two terms are found in a text the more likely they are related in the description of a particular concept. Proximity is used to increase the precision of a search.

SEARCH STATEMENT: COMPUTER OR PROCESSOR NOT MAINFRAME

SYSTEM OPERATION: Select all items discussing Computers and/or Processors that do not discuss Mainframes

If the terms COMPUTER and DESIGN are found within a few words of each other then the item is more likely to be discussing the design of computers than if the words are paragraphs apart.

The typical format for proximity is: TERM1 within "m" "units" of TERM2

The distance operator "m" is an integer number and units are in Characters, Words, Sentences, or Paragraphs. Certain items may have other semantic units that would prove useful in specifying the proximity operation. For very structured items, distances in characters prove useful. For items containing embedded images (e.g., digital photographs), text between the images could help in precision when the objective is in locating a certain image. Sometimes the proximity relationship contains a direction operator indicating the direction (before or after) that the second term must be found within the number of units specified. The default is either direction. A special case of the Proximity operator is the

Adjacent (ADJ) operator that normally has a distance operator of one and a forward only direction (i.e., in WAIS). Another special case is where the distance is set to zero meaning within the same semantic unit.

5.1.3 Contiguous Word Phrases

A Contiguous Word Phrase (CWP) is both a way of specifying a query term and a special search operator. A Contiguous Word Phrase is two or more words that are treated as a single semantic unit. An example of a CWP is "United States of America." It is four words that specify a search term representing a single specific semantic concept (a country). Thus a query could specify "manufacturing" AND "United States of America" which returns any item that contains the word "manufacturing" and the contiguous words "United States of America."

SEARCH STATEMENT: "Venetian" ADJ "Blind"

SYSTEM OPERATION: would find items that mention a Venetian Blind on a window but not items discussing a Blind Venetian

A contiguous word phrase also acts like a special search operator that is similar to the proximity (Adjacency) operator but allows for additional specificity. If two terms are specified, the contiguous word phrase and the proximity operator using directional one word parameters or the adjacent operator are identical. For contiguous word phrases of more than two terms the only way of creating an equivalent search statement using proximity and Boolean operators is via nested Adjacencies. This is because Proximity and Boolean operators are binary operators but contiguous word phrases are an "N"ary operator where "N" is the number of words in the CWP. Contiguous Word Phrases are called Literal Strings in WAIS and Exact Phrases in Retrieval Ware. In WAIS multiple Adjacency (ADJ) operators are used to define a Literal String (e.g., "United" ADJ "States" ADJ "of" ADJ "America").

5.1.4 Fuzzy Searches

Fuzzy Searches provide the capability to locate spellings of words that are similar to the entered search term. This function is primarily used to compensate for errors in spelling of words. Fuzzy searching increases recall at the expense of decreasing precision. In the process of expanding a query term fuzzy searching includes other terms that have similar spellings,

giving more weight to words in the database that have similar word lengths and position of the characters as the entered term.

A Fuzzy Search on the term “computer” would automatically include the following words from the information database: “computer,” “compiter,” “conputer,” “computter,” compute.”

A fuzzy search is done by means of a fuzzy matching program, which returns a list of results based on likely relevance even though search argument words and spellings may not exactly match. Exact and highly relevant matches appear near the top of the list. Subjective relevance ratings, usually as percentages, may be given. A fuzzy matching program can operate like a spell checker and spelling-error corrector. For example, if a user types "Misissippi" into Yahoo or Google, a list of hits is returned along with the question, "Did you mean Mississippi?" Alternative spellings, and words that sound the same but are spelled differently, are given. Fuzzy searching is especially useful when researching unfamiliar, foreign-language, or sophisticated terms, the proper spellings of which are not widely known. Fuzzy searching can also be used to locate individuals based on incomplete or partially inaccurate identifying information.

5.1.5 Term Masking

Term masking is the ability to expand a query term by masking a portion of the term and accepting as valid any processing token that maps to the unmasked portion of the term. There are two types of search term masking:

i) Fixed length
ii) Variable length

Fixed length masking is a single position mask. It masks out any symbol in a particular position or the lack of that position in a word. It not only allows any character in the masked position, but also accepts words where the position does not exist.

SEARCH STATEMENT: comput*

SYSTEM OPERATION: Matches “computers,” “computing,” , “computes”

Variable length “don’t cares” allows masking of any number of characters within a processing token. The masking may be in the front, at the end, at both front and end, or imbedded. The first three of these cases are called suffix search, prefix search and imbedded character string search, respectively. The use of an imbedded variable length don’t care is

seldom used. If "*" represents a variable length don't care then the following are examples of its use:

"*COMPUTER"	Suffix Search
"COMPUTER*"	Prefix Search
"*COMPUTER*"	Imbedded String Search

5.1.6 Numeric and Date Ranges

Term masking is useful when applied to words, but does not work for finding ranges of numbers or numeric dates. To find numbers larger than "125," using a term "125*" will not find any number except those that begin with the digits "125." Systems, as part of their normalization process, characterize words as numbers or dates. This allows for specialized numeric or date range processing against those words. A user could enter inclusive (e.g., "125-425" or "4/2/93- 5/2/95" for numbers and dates) to infinite ranges (">125," "<=233," representing "Greater Than" or "Less Than or Equal") as part of a query.

5.1.7 Concept/Thesaurus Expansion

The search terms can also be expanded via Thesaurus or Concept Class database reference tool. A Thesaurus is typically a one-level or two-level expansion of a term to other terms that are similar in meaning. An example for Thesaurus is shown in figure 1.1.

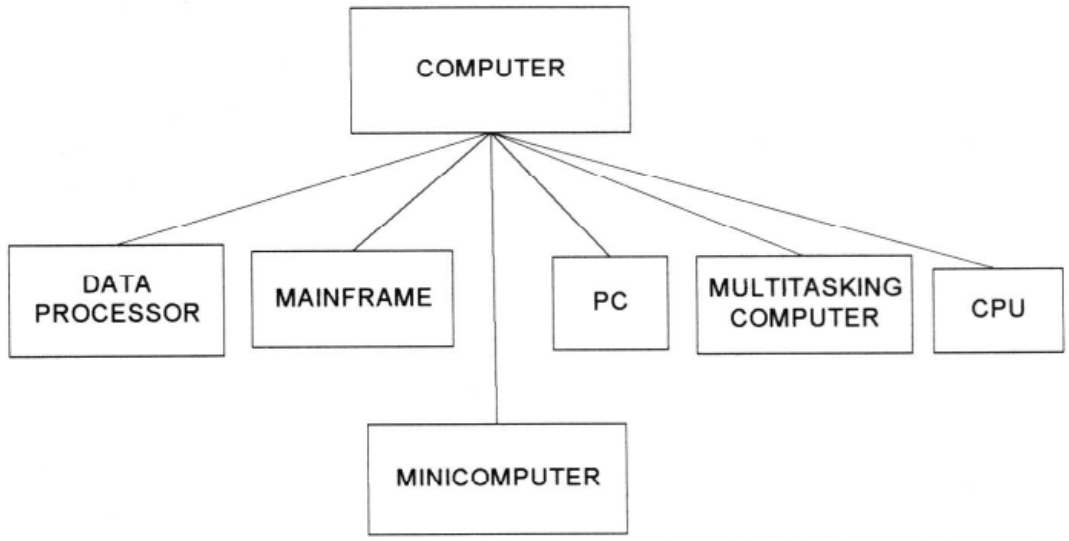


Figure 1.1 Thesaurus for term "computer"

A Concept Class is a tree structure that expands each meaning of a word into potential concepts that are related to the initial term (e.g., in the TOPIC system). Concept classes are sometimes implemented as a network structure that links word stems (e.g., in the RetrievalWare system). Concept class representations assist a user who has minimal knowledge of a concept domain by allowing the user to expand upon a particular concept showing related concepts. The following figure 1.2 illustrates the Concept Class.

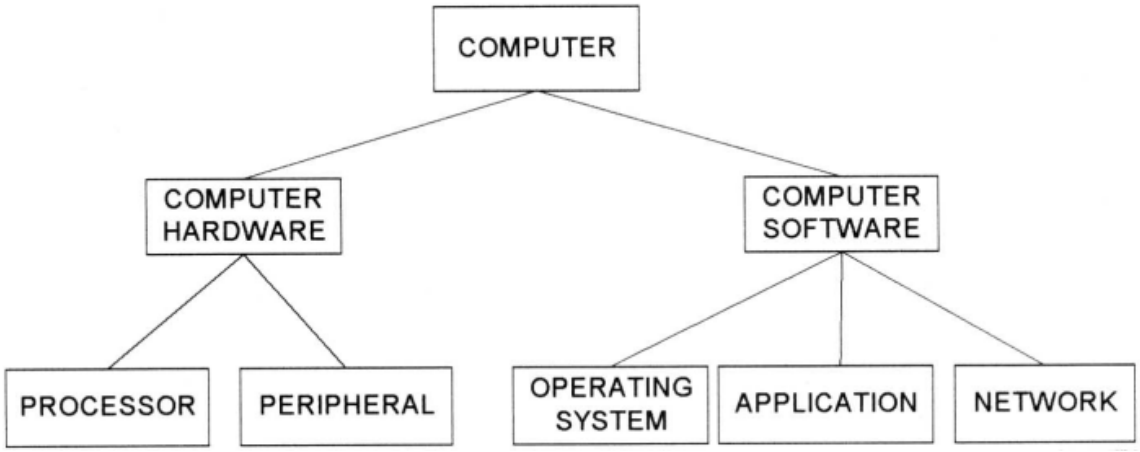


Figure 1.2 Hierarchical Concept Class Structure for "Computer"

Thesauri are either semantic or based upon statistics. A semantic thesaurus is a listing of words and then other words that are semantically similar. In executing a query, a term can be expanded to all related terms in the thesaurus or concept tree. Optionally, the user may display the thesaurus or concept tree and indicate which related terms should be used in a query. This function is essential to eliminate synonyms which introduce meanings that are not in the user's search statement. For example, a user searching on "pasture lands" and "fields" would not want all of the terms associated with "magnetic fields" included in the expanded search statement.

The problem with thesauri is that they are generic to a language and can introduce many search terms that are not found in the document database. An alternative uses the database or a representative sample of it to create statistically related terms. It is conceptually a thesaurus in that words that are statistically related to other words by their frequently occurring together in the same items. This type of thesaurus is much dependent upon the database being searched and may not be portable to other databases. Theoretically thesauri and concept trees could be used to either expand a search statement with additional terms or make it more

specific but substituting more specific terms. From this perspective expanding the terms increases the recall of the search with a possible decrease in precision.

5.1.8 Natural Language Queries

Natural Language Queries allow a user to enter a prose statement that describes the information that the user wants to find. The longer the prose, the more accurate the results returned. The most difficult logic case associated with Natural Language Queries is the ability to specify negation in the search statement and have the system to recognize it as negation. The system searches and finds those items most like the query statement entered.

An example of a Natural Language Query is:

What is the state of the art in text retrieval?

The system will search for:

state of the art AND text AND retrieval

Using the Natural Language Query search statement, a Boolean query also attempts to find the same information. A function is associated with natural language queries called as relevance feedback. The natural language does not have to be input by the user but just identified by the user. This introduces the concept of finding items that "are like" other items. Thus, a user could identify a particular item(s) in the database or text segments within item(s) and use that as the search statement.

5.1.9 Multimedia Queries

The current systems only focus on specification of still images as other search criteria. The still image could be used to search images that are part of an item. They also could be used to locate a specific scene in a video product. In the video modality, scene changes are extracted to represent changes in the information presentation. The scene changes are represented as a series of images. Additionally, where there is static text in the video, the current technology allows for OCRing the text.

The ability to search for audio as a match makes less sense as a user specification. To adequately perform the search, simulate the audio segment and then look for a match. Instead audio sources are converted to searchable text via audio transcription. This allows queries to be applied to the text. Thus the search algorithms must allow for errors in the data. The errors

are very different compared to OCR. OCR errors will usually create a text string that is not a valid word.

In automatic speech recognition (ASR), all errors are other valid words since ASR selects entries ONLY from a dictionary of words. Audio also allows the user to search on specific speakers, since speaker identification is relatively accurate against audio sources recognition (ASR), all errors are other valid words since ASR selects entries ONLY from a dictionary of words. Audio also allows the user to search on specific speakers, since speaker identification is relatively accurate against audio sources. The correlation between different parts of a query against different modalities is usually based upon time or location.

The correlation between different parts of a query against different modalities is usually based upon time or location. The most common example would be on time. For example if a video news program has been indexed, the user could have access to the scene changes, the transcribed audio, the closed captioning and the index terms that a user has assigned while displaying the video. The query could be "Find where Bill Clinton is discussing Cuban refugees and there is a picture of a boat". All of the separate tracks of information are correlated on a time basis. The system would return those locations where Bill Clinton is identified as the speaker (user the audio track and speaker identification), where in any of the text streams (OCRed text from the video, transcribed audio, closed captioning, or index terms) there is discussion of refugees and Cuba, and finally during that time segment there is at least one scene change that includes a boat.

5.2 Browse Capabilities

Once the search is complete, Browse capabilities provide the user with the capability to determine which items are of interest and select those to be displayed. There are two ways of displaying a summary of the items that are associated with a query: line item status and data visualization. From these summary displays, the user can select the specific items and zones within the items for display. The system also allows for easy transitioning between the summary displays and review of specific items. If searches resulted in high precision, then the importance of the browse capabilities would be lessened. Since searches return many items that are not relevant to the user’s information need, browse capabilities can assist the user in focusing on items that have the highest likelihood in meeting his need.

5.2.1 Ranking

The ranking is based upon predicted relevance values. The status summary displays the relevance score associated with the item along with a brief descriptor of the item. The score is an estimate of the search system on how closely the item satisfies the search statement. Typically relevance scores are normalized to a value between 0.0 and 1.0. The highest value of 1.0 is interpreted that the system is sure that the item is relevant to the search statement. This allows the user to determine at what point to stop reviewing items because of reduced likelihood of relevance.

The ranking based upon the characteristics of the item and the database, in many circumstances collaborative filtering is providing an option for selecting and ordering output. In this case, users when reviewing items provide feedback to the system on the relative value of the item being accessed. The system accumulates the various user rankings and uses this information to order the output for other user queries that are similar. Collaborative filtering has been very successful in sites such as AMAZON.COM MovieFinder.com, and CDNow.com in deciding what products to display to users based upon their queries. Some systems create relevance categories and indicate, by displaying items in different colors, which category an item belongs to. Other systems uses a nomenclature such as High, Medium High, Medium, Low, and Non-relevant.

5.2.2 Zoning

When the user displays a particular item, the user wants to see the minimum information needed to determine if the item is relevant. Once the determination is made an item is possibly relevant, the user wants to display the complete item for detailed review. Limited display screen sizes require selectability of what portions of an item a user needs to see to make the relevance determination.

For example, display of the Title and Abstract may be sufficient information for a user to predict the potential relevance of an item.

Limiting the display of each item to these two zones allows multiple items to be displayed on a single display screen. This makes maximum use of the speed of the user's cognitive process

in scanning the single image and understanding the potential relevance of the multiple items on the screen.

5.2.3 Highlighting

Highlighting is the display aid that indicates why an item was selected. Highlighting indicates the user quickly to focus on the relevant parts of the text to scan for item relevance. Different strengths of highlighting indicate how strongly the highlighted word participated in the selection of the item. Most systems allow the display of an item to begin with the first highlight within the item and allow subsequent jumping to the next highlight.

Another capability, which is gaining strong acceptance, is for the system to determine the passage in the document most relevant to the query and position the browse to start at that passage. Highlighting has always been useful in Boolean systems to indicate the cause of the retrieval. This is because of the direct mapping between the terms in the search and the terms in the item. Using Natural Language Processing, automatic expansion of terms via thesauri, and the similarity ranking algorithms discussed in detail later in this book, highlighting loses some of its value The terms being highlighted that caused a particular item to be returned may not have direct or obvious mapping to any of the search terms entered. This causes frustration by the user trying to guess why a particular item was retrieved and how to use that information in reformulating the search statement to make it more exact. In a ranking system different terms can contribute to different degrees to the decision to retrieve an item. The highlighting may vary by introducing colors and intensities to indicate the relative importance of a particular word in the item in the decision to retrieve the item.

5.3 Miscellaneous Capabilities

There are many additional functions that facilitate the user's ability to input queries, reducing the time it takes to generate the queries, and reducing a priori the probability of entering a poor query. Vocabulary browse provides knowledge on the processing tokens available in the searchable database and their distribution in terms of items within the database. Iterative searching and search history logs summarize previous search activities by the user allowing access to previous results from the current user session. Canned queries allow access to queries generated and saved in previous user sessions.

5.3.1 Vocabulary Browse

Vocabulary Browse provides the capability to display in alphabetical sorted order words from the document database. Logically, all unique words in the database are kept in sorted order along with a count of the number of unique items in which the word is found. The user can enter a word or word fragment and the system will begin to display the dictionary around the entered text. The system indicates what word fragment the user entered and then alphabetically displays other words found in the database in collating sequence on either side of the entered term. The user can continue scrolling in either direction reviewing additional terms in the database. Vocabulary browse provides information on the exact words in the database. It helps the user determine the impact of using a fixed or variable length mask on a search term and potential mis-spellings. The user can determine that entering the search term "compul*" in effect is searching for "compulsion" or "compulsive" or "compulsory." It also shows that someone probably entered the word "computen" when they really meant "computer." It provides insight on the impact of using terms in a search. By vocabulary browsing, a term may be seen to exist in a large number of documents which could make it a poor candidate as an ORed term requiring additional ANDed terms to focus on items of interest. The search term "computer" would return an excessive number of hits if used as an "OR" term.

5.3.2 Iterative Search and Search History Log

Take the original query and add additional search statement against it in an AND condition. This process of refining the results of a previous search to focus on relevant items is called iterative search. This also applies when a user uses relevance feedback to enhance a previous search. During a login session, a user could execute many queries to locate the needed information. To facilitate locating previous searches as starting points for new searches, search history logs are available. The search history log is the capability to display all the previous searches that were executed during the current session. The query along with the search completion status showing number of hits is displayed.

5.3.3 Canned Query

Canned queries are queries that tend to be quite common and are stored in such a way that they can be easily executed without having to re-enter the details of the query each time. The

capability to name a query and store it to be retrieved and executed during a later user session is called canned or stored queries. Users tend to have areas of interest within which they execute their searches on a regular basis. A canned query allows a user to create and refine a search that focuses on the user's general area of interest one time and then retrieve it to add additional search criteria to retrieve data that is currently needed. For example, if you had a database with softball team stats in it, you could create canned queries for each team in the database so that a user would only have to click on a link to "Blue Team Stats" in order to bring up all of the data stored about the Green Team.

5.3.4 Multimedia

Once lists of potential items that satisfy the query are discovered, the techniques for displaying them when they are multimedia introduce new challenges. The challenge is to present the most information possible to the user and allow the user to select the items to be retrieved. To display more aggregate data, textual interfaces sometimes allow for clustering of the hits and then use of graphical display to show a higher level view of the information. Neither of these techniques tends themselves well when the information is multimodal. The textual aspect of the multimedia can be used to apply all of the techniques described above. The transcribed audio becomes a critical augmentation in users reviewing audio files. Thus in addition to listening to the audio, the user can visually be following the transcribed text. This provides a mechanism for the user to perceive the context and additionally provides a quick scanning option to look ahead at upcoming information to be used in conjunction with the audio processing of the original source.

5.4 Summary

This unit provides an overview of the functions commonly associated with Information Retrieval Systems. The functions namely, the search and browse capabilities which are very essential in locating the area of interest. There are many additional functions such as vocabulary browse, canned queries, iterative search and history logs, multimedia that facilitate the user's ability to input queries, reducing the time it takes to generate the queries, and reducing a priori the probability of entering a poor query.

5.5 Keywords

Boolean Logic, Proximity, Contiguous Word Phrases, Fuzzy Searches, Term Masking Concept and Thesaurus Expansions, Natural Language Queries, Multimedia Queries, Ranking, Zoning, Highlighting, Vocabulary Browse, Iterative Search, History Log, Canned Query, Multimedia

5.6 Questions

1. Describe the rationale why use of proximity will improve precision versus use of just the Boolean functions. Discuss its effect on improvement of recall.
2. Show that the proximity function cannot be used to provide an equivalent to a Contiguous Word Phrase.
3. What are the similarities and differences between use of fuzzy searches and term masking? What are the potentials for each to introduce errors?
4. Are thesauri a subclass of concept classes? Justify your answer.
5. Which would users prefer, Boolean queries or Natural Language queries? Why?
6. Ranking is one of the most important concepts in Information Retrieval Systems. What are the difficulties in applying ranking when Boolean queries are used?

5.7 REFERENCES FOR FURTHER READING

Aho, A., B. Kernighan, and P. Weinberger. 1988. *The AWK Programming Language.* Reading, Mass.: Addison-Wesley.

Belkin N. J., and W. B. Croft. 1987. "Retrieval Techniques," in *Annual Review of Information Science and Technology*, ed. M. Williams. New York: Elsevier Science Publishers, 109-145.

Faloutsos, C. 1985. "Access Methods for Text," *Computing Surveys*, 17(1), 49-74.

Frakes, W. B. 1984. "Term Conflation for Information Retrieval," in *Research and Development in Information Retrieval*, ed. C. S. van Rijsbergen. Cambridge: Cambridge University Press.

Prieto-Diaz, R., and G. Arango. 1991. *Domain Analysis: Acquisition of Reusable Information for Software Construction*. New York: IEEE Press.

Salton, G., and M. Mcgill 1983. *An Introduction to Modern Information Retrieval*. New York: McGraw-Hill.

Sedgewick, R. 1990. *Algorithms in C*. Reading, Mass.: Addison-Wesley.

Sparck-Jones, K. 1981. *Information Retrieval Experiment*. London: Butterworths.

Tong, R, ed. 1989. Special Issue on Knowledge Based Techniques for Information Retrieval, *International Journal of Intelligent Systems*, 4(3).

Van Rijsbergen, C. J. 1979. *Information Retrieval*. London: Butterworths

UNIT 6: CATALOGING AND INDEXING

Structure

6.0 INTRODUCTION

One of the most critical aspects of an information system that determines its effectiveness is how it represents concepts in items. The transformation from the received item to the searchable data structure is called Indexing. This process can be manual or automatic, creating the basis for direct search of items in the Document Database or indirect search via Index Files. Rather than trying to create a searchable data structure that directly maps to the text in the input items, some systems transform the item into a completely different representation that is concept based and use this as the searchable data structure. Once the searchable data structure has been created, techniques must be defined that correlate the query statement to the set of items in the database to determine the items to be returned to the user.

6.1 History and Objectives of Indexing

To understand the system design associated with creation and manipulation of the searchable data structures, it is necessary to understand the objectives of the indexing process. Reviewing the history of indexing shows the dependency of information processing capabilities on manual and then automatic processing systems.

6.1.1 History

Indexing (originally called Cataloging) is the oldest technique for identifying the contents of items to assist in their retrieval. The objective of cataloging is to give access points to a collection that are expected and most useful to the users of the information.

As early as the third-millennium, in Babylon, libraries of cuneiform tablets were arranged by subject (Hyman-89). Up to the 19th Century there was little advancement in cataloging, only changes in the methods used to represent the basic information (Norris-69). In the late 1800s subject indexing became hierarchical (e.g., Dewey Decimal System). In 1963 the Library of Congress initiated a study on the computerization of bibliographic surrogates. From 1966 - 1968 the Library of Congress ran its MARC I pilot project. MARC (MAchine Readable Cataloging) standardizes the structure, contents and coding of bibliographic records.

The earliest commercial cataloging system is DIALOG, which was developed by Lockheed Corporation in 1965 for NASA. It became commercial in 1978 with three government files of indexes to technical publications. By 1988, when it was sold to Knight-Ridder, DIALOG contained over 320 index databases used by over 91,000 subscribers in 86 countries (Harper-81).

Indexing (cataloging), until recently, was accomplished by creating a bibliographic citation in a structured file that references the original text. These files contain citation information about the item, key wording the subject(s) of the item and, in some systems a constrained length free text field used for an abstract/summary. The indexing process is typically performed by professional indexers associated with library organizations.

The initial introduction of computers to assist the cataloguing function did not change its basic operation of a human indexer determining those terms to assign to a particular item.

In the 1990s, the significant reduction in cost of processing power and memory in modern computers, along with access to the full text of an item from the publishing stages in electronic form, allow use of the full text of an item as an alternative to the indexer-generated subject index.

6.1.2 Objectives

The objectives of indexing have changed with the evolution of Information Retrieval Systems. Availability of the full text of the item in searchable form alters the objectives historically used in determining guidelines for manual indexing.

The full text searchable data structure for items in the Document File provides a new class of indexing called total document indexing. In this environment, all of the words within the item are potential index descriptors of the subject(s) of the item.

The availability of items in electronic form changes the objectives of manual indexing. The source information (frequently called citation data) can automatically be extracted. There is still some utility to the use of indexes for index term standardization. Modern systems, with the automatic use of thesauri and other reference databases, can account for diversity of language/vocabulary use and thus reduce the need for controlled vocabularies.

The words used in an item do not always reflect the value of the concepts being presented. It is the combination of the words and their semantic implications that contain the value of the concepts being discussed. The utility of a concept is also determined by the user's need.

In addition to the primary objective of representing the concepts within an item to facilitate the user's finding relevant information, electronic indexes to items provide a basis for other applications to assist the user.

6.2 INDEXING PROCESS

When an organization with multiple indexers decides to create a public or private index some procedural decisions on how to create the index terms assist the indexers and end users in knowing what to expect in the index file.

The Decisions are:

1) The scope of the indexing to define what level of detail the subject index will contains. This is based upon usage scenarios of the end users.
2) The need to link index terms together in a single index for a particular concept.

Linking index terms is needed when there are multiple independent concepts found within an item.

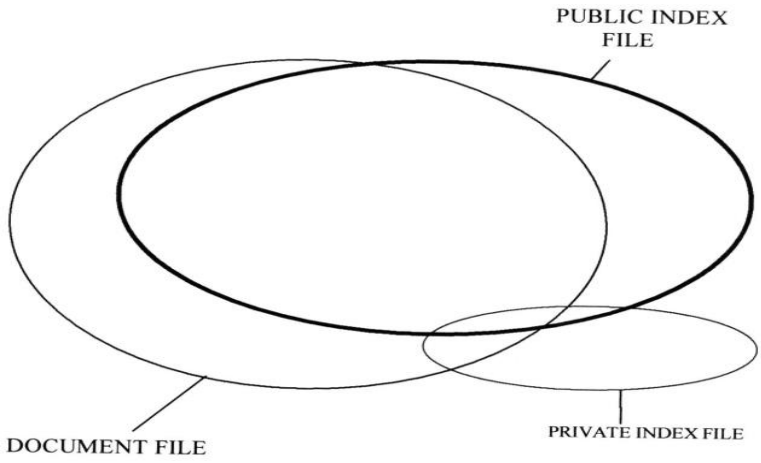


Figure 6.1: Items Overlap between Full Item Indexing, Public File Indexing and Private File Indexing

6.2.1 Scope of Indexing

When performed manually, the process of reliably and consistently determining the bibliographic terms that represent the concepts in an item is extremely difficult. Problems arise from interaction of two sources: the author and the indexer. The vocabulary domain of the author may be different than that of the indexer, causing the indexer to misinterpret the emphasis and possibly even the concepts being presented. The indexer is not an expert on all areas and has different levels of knowledge in the different areas being presented in the item.

This results in different quality levels of indexing. The indexer must determine when to stop the indexing process.

There are two factors involved in deciding on what level to index the concepts in an item:

1) The exhaustivity: Exhaustivity of indexing is the extent to which the different concepts in the item are indexed.
2) The specificity: Specificity relates to the preciseness of the index terms used in indexing.

This approach requires a minimal number of index terms per item and reduces the cost of generating the index. For example, indexing this paragraph would only use the index term "indexing." High exhaustivity and specificity indexes almost every concept in the item using as many detailed terms as needed. Low exhaustivity has an adverse effect on both precision and recall. If the full text of the item is indexed, then low exhaustivity is used to index the abstract concepts not explicit in the item with the expectation that the typical query searches both the index and the full item index.

Another decision on indexing is what portions of an item should be indexed. The simplest case is to limit the indexing to the Title or Title and Abstract zones.

Weighting of index terms is not common in manual indexing systems. Weighting is the process of assigning an importance to an index term's use in an item. The weight should represent the degree to which the concept associated with the index term is represented in the item.

6.2.2 Pre-coordination and Linkages

Another decision on the indexing process is whether linkages are available between index terms for an item. Linkages are used to correlate related attributes associated with concepts discussed in an item. This process of creating term linkages at index creation time is called precoordination. When index terms are not coordinated at index time, the coordination occurs at search time. This is called post coordination that is coordinating terms after (post) the

indexing process. Post coordination is implemented by "AND"ing index terms together, which only find indexes that, have all of the search terms.

Factors that must be determined in the linkage process are the number of terms that can be related, any ordering constraints on the linked terms, and any additional descriptors are associated with the index terms (Vickery-70). The range of the number of index terms that can be linked is not a significant implementation issue and primarily affects the design of the indexer's user interface. When multiple terms are being used, the possibility exists to have relationships between the terms.

For example, the capability to link the source of a problem, the problem and who is affected by the problem may be desired. Each term must be caveated with one of these three categories along with linking the terms together into an instance of the relationships describing one semantic concept. The order of the terms is one technique for providing additional role descriptor information on the index terms. Use of the order of the index terms to implicitly define additional term descriptor information limits the number of index terms that can have a role descriptor. If order is not used, modifiers may be associated with each term linked to define its role. This technique allows any number of terms to have the associated role descriptor.

6.3 AUTOMATIC INDEXING

Automatic indexing is the capability for the system to automatically determine the index terms to be assigned to an item. The simplest case is when all words in the document are used as possible index terms (total document indexing).

More complex processing is required when the objective is to emulate a human indexer and determine a limited number of index terms for the major concepts in the item. As discussed, the advantages of human indexing are the ability to determine concept abstraction and judge the value of a concept. The disadvantages of human indexing over automatic indexing are cost, processing, time and consistency. Automatic indexing requires only a few seconds or less of computer time based upon the size of the processor and the complexity of the algorithms to generate the index. Another advantage to automatic indexing is the predictably

of algorithms. If the indexing is being performed automatically, by an algorithm, there is consistency in the index term selection process. Human indexers typically generate different indexing for the same document.

Indexes resulting from automated indexing fall into two classes:

i) Weighted indexing system

ii) Unweighted indexing system

In a weighted indexing system, an attempt is made to place a value on the index term's representation of its associated concept in the document. An index term's weight is based upon a function associated with the frequency of occurrence of the term in the item. The query process uses the weights along with any weights assigned to terms in the query to determine a scalar value (rank value) used in predicting the likelihood that an item satisfies the query.

In an unweighted indexing system, the existence of an index term in a document and sometimes its word location(s) are kept as part of the searchable data structure. No attempt is made to discriminate between the values of the index terms in representing concepts in the item. Looking at the index, it is not possible to tell the difference between the main topics in the item and a casual reference to a concept. Queries against unweighted systems are based upon Boolean logic and the items in the resultant Hit file are considered equal in value.

Automatic indexing can either try to preserve the original text of an item basing the final set of searchable index values on the original text or map the item into a completely different representation, called concept indexing, and use the concepts as a basis for the final set of index values.

6.3.1 Indexing by Term

There are two major techniques for creation of the index: statistical and natural language. Statistical techniques can be based upon vector models and probabilistic models with a special case being Bayesian models. They are classified as statistical because their calculation of weights uses statistical information such as the frequency of occurrence of words and their

distributions in the searchable database. Natural language techniques also use some statistical information, but perform more complex parsing to define the final set of index concepts.

The model that has been most successful in this area is the Bayesian approach. This approach is natural to information systems and is based upon the theories of evidential reasoning. Bayesian approaches have long been applied to information systems. The Bayesian approach could be applied as part of index term weighting, but usually is applied as part of the retrieval process by calculating the relationship between an item and a specific query. A Bayesian network is a directed acyclic graph in which each node represents a random variable and the arcs between the nodes represent a probabilistic dependence between the node and its parents. Figure shows the basic weighting approach for index terms or associations between query terms and index terms.

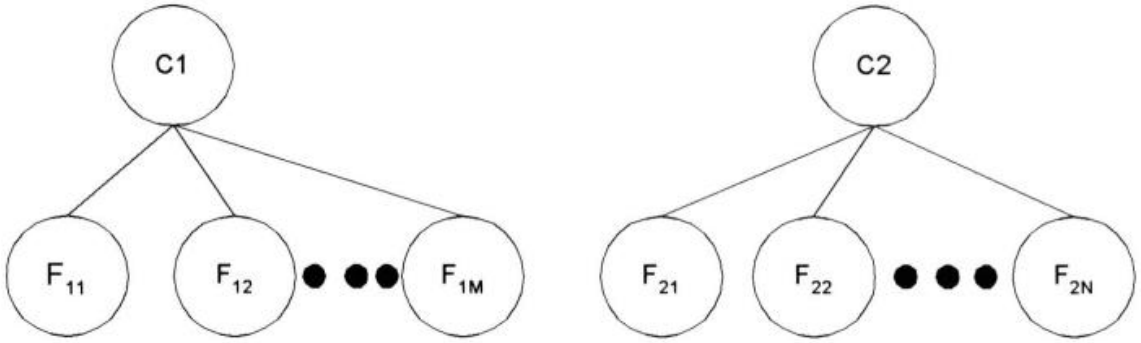


Figure 6.2 Two-level Bayesian network

The nodes $\mathbf{C_1}$ and $\mathbf{C_2}$ represent "the item contains $\mathbf{C_i}$" and the F nodes represent "the item has feature (e.g., words)$\mathbf{F_{ij}}$". The network could also be interpreted as C representing concepts in a query and F representing concepts in an item. The goal is to calculate the probability of $\mathbf{C_i}$ given $\mathbf{F_{ij}}$. To perform that calculation two sets of probabilities are needed:

1. The prior probability $\mathbf{P\ (C_i)}$that an item is relevant to concept C

2. The conditional probability $\mathbf{P\ (F_{ij}/C_i)}$ that the features $\mathbf{F_{ij}}$ where $\mathbf{j= m}$ are present in an item given that the item contains topic

$$\mathbf{P(C_i/F_{i1}, \dots , F_{im}) = P(C_i)\ P(F_{i1}, \dots , F_{im}/C_i)\backslash P(F_{i1}, \dots , F_{im}).}$$

If the goal is to provide ranking as the result of a search by the posteriors, the Bayes rule can be simplified to a linear decision rule:

$$\mathbf{g(C_i/F_{i1, \dots ,} F_{im}) = \Sigma_k\ I(F_{ik})\ w(F_{ik},\ C_i)}$$

$I(F_{ik})$ Where is an indicator variable that equals F_{ik} 1 only if is present in the item and w is a coefficient corresponding to a specific feature/concept pair. A careful choice of w produces a ranking in decreasing order that is equivalent to the order produced by the posterior probabilities. Interpreting the coefficients, w, as weights corresponding to each feature (e.g., index term) and the function g as the sum of the weights of the features, the result of applying the formula is a set of term weights.

Another approach to defining indexes to items is via use of natural language processing. The DR-LINK system processes items at the morphological, lexical, semantic, syntactic, and discourse levels. Each level uses information from the previous level to perform its additional analysis. The discourse level is abstracting information beyond the sentence level and can determine abstract concepts using pre-defined models of event relationships. This allows the indexing to include specific term as well as abstract concepts. Normal automatic indexing does a poor job at identifying and extracting "verbs" and relationships between objects based upon the verbs.

6.3.2 Indexing by Concept

The basis for concept indexing is that there are many ways to express the same idea and increased retrieval performance comes from using a single representation. Indexing by term treats each of these occurrences as a different index and then uses thesauri or other query expansion techniques to expand a query to find the different ways the same thing has been represented. Concept indexing determines a canonical set of concepts based upon a test set of terms and uses them as a basis for indexing all items. This is also called Latent Semantic Indexing because it is indexing the latent semantic information in items. The determined set of concepts does not have a label associated with each concept, but is a mathematical representation.

Word stems, items and queries are represented by high dimensional (at least 300 dimensions) vectors called context vectors. Each dimension in a vector could be viewed as an abstract concept class. The approach is based upon cognitive science work by Waltz and Pollack. To define context vectors, a set of *n* features are selected on an ad hoc basis (e.g., high frequency terms after removal of stop words). The selection of the initial features is not critical since

they evolve and expand to the abstract concept classes used in the indexing process. For any word stem *k*, its context vector V*k*is an *n*-dimensional vector with each component *j* interpreted as follows:

$\mathbf{V}^k$ positive if *k* is strongly associated with feature *j*

$\mathbf{V}^k \approx \mathbf{0}$ if word *k* is not associated with feature *j*

$\mathbf{V}^k$ negative if word *k* contradicts feature *j*

The interpretation of components for concept vectors is exactly the same as weights in neural networks. Each of the *n* features is viewed as an abstract concept class. Then each word stem is mapped to how strongly it reflects each concept in the items in the corpus. There is overlap between the concept classes providing a distributed representation and insulating against a small number of entries for context vectors that could have no representation for particular stems, once the context vectors for stems are determined, they are used to create the index for an item. A weighted sum of the context vectors for all the stems in the item is calculated and normalized to provide a vector representation of the item in terms of the *n* concept classes (features). Queries (natural language only) go through the same analysis to determine vector representations. These vectors are then compared to the item vectors.

6.3.3 Multimedia Indexing

Indexing video or images can be accomplished at the raw data level (e.g., the aggregation of raw pixels), the feature level distinguishing primitive attributes such as color and luminance, and at the semantic level where meaningful objects are recognized (e.g., an airplane in the image/video frame). An example is processing of video. The system (e.g., Virage) will periodically collect a frame of video input for processing. It might compare that frame to the last frame captured to determine the differences between the frames. If the difference is below a threshold it will discard the frame. For a frame requiring processing, it will define a vector that represents the different features associated with that frame. Each dimension of the vector represents a different feature level aspect of the frame. The vector then becomes the unit of processing in the search system. This is similar to processing an image. Semantic level indexing requires pattern recognition of objects within the images.

In addition to storing the extracted index searchable data, a multimedia item needs to also store some mechanism to correlate the different modalities during search. There are two main mechanisms that are used, positional and temporal. Positional is used when the modalities are interspersed in a linear sequential composition. For example a document that has images or audio inserted can be considered a linear structure and the only relationship between the modalities will be the juxtaposition of each modality. This would allow for a query that would specify location of an image of a boat within one paragraph of "Cuba and refugees".

The second mechanism is based upon time because the modalities are executing concurrently. The typical video source off television is inherently a multimedia source. It contains video, audio, and potentially closed captioning. Also the creations of multimedia presentations are becoming more common using the Synchronized Multimedia Integration Language (SMIL). It is a mark-up language designed to support multimedia presentations that integrate text (e.g., from slides or free running text) with audio, images and video.

6.4 INFORMATION EXTRACTION

There are two processes associated with information extraction:

1. Determination of facts to go into structured fields in a database and
2. Extraction of text that can be used to summarize an item.

In the first case only a subset of the important facts in an item may be identified and extracted. In summarization all of the major concepts in the item should be represented in the summary. The process of extracting facts to go into indexes is called Automatic File Build; its goal is to process incoming items and extract index terms that will go into a structured database. This differs from indexing in that its objective is to extract specific types of information versus understanding all of the text of the document.

An Information Retrieval System’s goal is to provide an in-depth representation of the total contents of an item. An Information Extraction system only analyzes those portions of a document that potentially contain information relevant to the extraction criteria.

The objective of the data extraction is in most cases to update a structured database with additional facts. The updates may be from a controlled vocabulary or substrings from the item as defined by the extraction rules. The term “slot” is used to define a particular category of information to be extracted. Slots are organized into templates or semantic frames. Information extraction requires multiple levels of analysis of the text of an item. It must understand the words and their context.

In establishing metrics to compare information extraction, the previously defined measures of precision and recall are applied with slight modifications to their meaning. Recall refers to how much information was extracted from an item versus how much should have been extracted from the item. It shows the amount of correct and relevant data extracted versus the correct and relevant data in the item. Precision refers to how much information was extracted accurately versus the total information extracted.

Additional metrics used are over generation and fallout. Over generation measures the amount of irrelevant information that is extracted. This could be caused by templates filled on topics that are not intended to be extracted or slots that get filled with non-relevant data. Fallout measures how much a system assigns incorrect slot fillers as the number of potential incorrect slot fillers increases.

These measures are applicable to both human and automated extraction processes. Human beings fall short of perfection in data extraction as well as automated systems. The best source of analysis of data extraction is from the Message Understanding Conference Proceedings.

Another related information technology is document summarization. Rather than trying to determine specific facts, the goal of document summarization is to extract a summary of an item maintaining the most important ideas while significantly reducing the size.

Examples of summaries that are often part of any item are titles, table of contents, and abstracts with the abstract being the closest. The abstract can be used to represent the item for search purposes or as a way for a user to determine the utility of an item without having to read the complete item. It is not feasible to automatically generate a coherent narrative summary of an item with proper discourse, abstraction and language usage.

Restricting the domain of the item can significantly improve the quality of the output. The more restricted goals for much of the research is in finding subsets of the item that can be extracted and concatenated and represents the most important concepts in the item. There is no guarantee of readability as a narrative abstract and it is seldom achieved. It has been shown that extracts of approximately 20 per cent of the complete item can represent the majority of significant concepts. Different algorithms produce different summaries.

Most automated algorithms approach summarization by calculating a score for each sentence and then extracting the sentences with the highest scores. Some examples of the scoring techniques are use of rhetorical relations, contextual inference and syntactic coherence using cue words, term location and statistical weighting properties.

6.5 SUMMARY

This chapter introduces the concepts behind indexing. Historically, term indexing was applied to a human-generated set of terms that could be used to locate an item. With the advent of computers and the availability of text in electronic form, alternatives to human indexing are available and essential. There is too much information in electronic form to make it feasible for human indexing of each item. Thus automated indexing techniques are absolutely essential. When humans performed the indexing, there were guidelines on the scope of the indexing process. They were needed to ensure that the human indexers achieved the objectives of a particular indexing effort.

Automated indexing systems try to achieve these by using weighted and natural language systems and by concept indexing. The goal of automatic indexing is not to achieve equivalency to human processing, but to achieve sufficient interpretation of items to allow users to locate needed information with the minimum amount of wasted effort.

The focus of text summarization is still on just the location of text segments that adequately represent an item. The combining of these segments into a readable “abstract” is still an unachievable goal. In the near term, a summarization that may not be grammatically correct

but adequately covers the concepts in an item can be used by user to determine if the complete item should be read in detail.

The original text of items is not being searched. The extracted index information is realistically the only way to find information. The weaker the theory and implementation of the indexing algorithms is, the greater the impact on the user in wasting energy to find needed information.

6.6 KEYWORDS

Indexing, Pre-coordination and linkages, Concept Indexing, Term indexing, Multimedia indexing, Information Extraction

6.7 QUESTIONS

1. Under what circumstances is manual indexing not required to ensure finding Information? Postulate an example where this is true.
2. Does high specificity always imply high exhaustivity? Justify your answer.
3. Trade off the use of pre coordination versus post coordination.
4. What are the problems with Luhn's concept of "resolving power"?
5. How does the process of information extraction differ from the process of document indexing?

6.8 REFERENCES FOR FURTHER READING

1 Belkin N. J., and W. B. Croft. 1987. "Retrieval Techniques," in *Annual Review of Information Science and Technology*, ed. M. Williams. New York: Elsevier Science Publishers, 109-145.

2 Faloutsos, C. 1985. "Access Methods for Text," *Computing Surveys*, 17(1), 49-74.

3 Frakes, W. B. 1984. "Term Conflation for Information Retrieval," in *Research and Development in Information Retrieval*, ed. C. S. van Rijsbergen. Cambridge: Cambridge University Press.

4 Prieto-Diaz, R., and G. Arango. 1991. *Domain Analysis: Acquisition of Reusable Information for Software Construction*. New York: IEEE Press.

5 Salton, G., and M. Mcgill 1983. *An Introduction to Modern Information Retrieval*. New York: McGraw-Hill.

6 Sparck-Jones, K. 1981. *Information Retrieval Experiment*. London: Butterworths.

7 Tong, R, ed. 1989. Special Issue on Knowledge Based Techniques for Information Retrieval, *International Journal of Intelligent Systems*, 4(3).

8 Van Rijsbergen, C. J. 1979. *Information Retrieval*. London: Butterworths

UNIT 7: DATA STRUCTURES FOR INFORMATION RETREIVAL

Structure

7.0 INTRODUCTION TO DATA STRUCTURE

There are usually two major data structures in any information system. One structure stores and manages the received items in their normalized form. The process supporting this structure is called the "document manager." The other major data structure contains the processing tokens and associated data to support search. The results of a search are references to the items that satisfy the search statement, which are passed to the document manager for retrieval.

The most common data structure encountered in both data base and information systems is the inverted file system. It minimizes secondary storage access when multiple search terms are applied across the total database. All commercial and most academic systems use inversion as the searchable data structure. A variant of the searchable data structure is the N-gram structure that breaks processing tokens into smaller string units (which is why it is sometimes discussed under stemming) and uses the token fragments for search. N-grams have demonstrated improved efficiencies and conceptual manipulations over full word inversion.

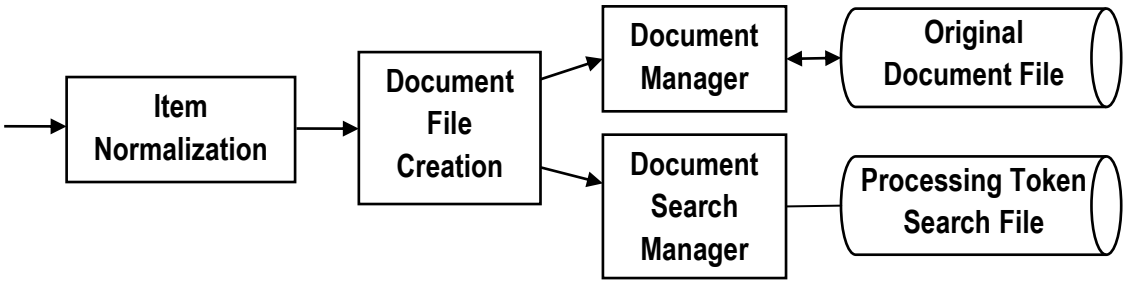


Figure 7.1: Major Data Structures

7.1 INVERTED FILE STRUCTURE

The most common data structure used in both database management and Information Retrieval Systems is the inverted file structure. Inverted file structures are composed of three basic files:

a) the document file,
b) the inversion lists (sometimes called posting files) and
c) the dictionary (or index file).

The name "inverted file" comes from its underlying methodology of storing an inversion of the documents: inversion of the document from the perspective that, for each key word (or attribute), a list of documents in which the key word is found in is stored (the inversion list for that key word). Each document in the system is given a unique numerical identifier. It is that identifier that is stored in the inversion list.

The concept of the inverted file type of index is as follows. Assume a set of documents. Each document is assigned a list of keywords or attributes, with optional relevance weights associated with each keyword. An inverted file is then the sorted list (or index) of keywords, with each keyword having links to the documents containing that keyword (see Figure 7.1). This is the kind of index found in most commercial library systems.

The use of an inverted file improves search efficiency by several orders of magnitude, a necessity for very large text files. The penalty paid for this efficiency is the need to store a data structure that ranges from 10 percent to 100 percent or more of the size of the text itself, and a need to update that index as the data set changes.

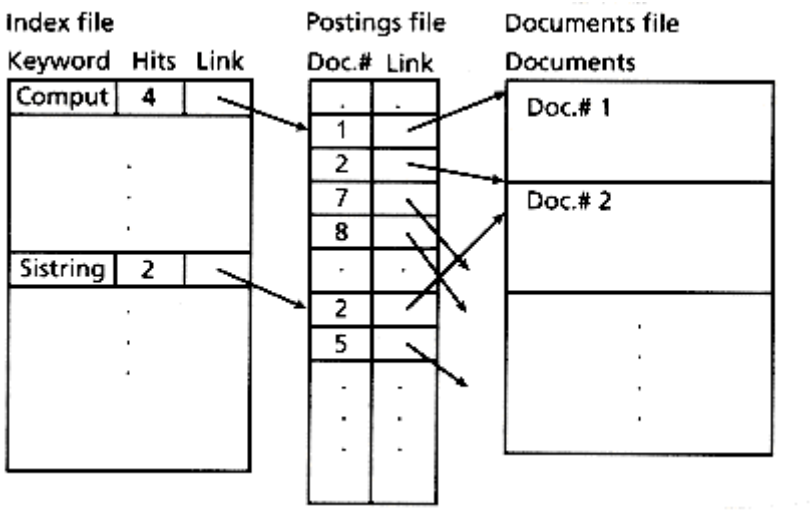


Figure 7.2: An inverted file implemented using a sorted array

Restrictions: Usually there are some restrictions imposed on these indices and consequently on later searches. Examples of these restrictions are:

- a controlled vocabulary which is the collection of keywords that will be indexed. Words in the text that are not in the vocabulary will not be indexed, and hence are not searchable.
- a list of stop words (articles, prepositions, etc.) that for reasons of volume or precision and recall will not be included in the index, and hence are not searchable.
- a set of rules that decide the beginning of a word or a piece of text that is indexable. These rules deal with the treatment of spaces, punctuation marks, or some standard prefixes, and may have significant impact on what terms are indexed.
- a list of character sequences to be indexed (or not indexed). In large text databases, not all character sequences are indexed; for example, character sequences consisting of all numerics are often not indexed.

It should be noted that the restrictions that determine what is to be indexed are critical to later search effectiveness and therefore these rules should be carefully constructed and evaluated.

Advantages: Inversion lists structures are used because they provide optimum performance in searching large databases. The optimality comes from the minimization of data flow in resolving a query. Only data directly related to the query are retrieved from secondary storage. Also there are many techniques that can be used to optimize the resolution of the query based upon information maintained in the dictionary.

Inversion list file structures are well suited to store concepts and their relationships. Each inversion list can be thought of as representing a particular concept. The inversion list is then a concordance of all of the items that contain that concept. Finer resolution of concepts can additionally be maintained by storing locations with an item and weights of the item in the inversion lists. With this information, relationships between concepts can be determined as part of search algorithms. Location of concepts is made easy by their listing in the dictionary and inversion lists. For Natural Language Processing algorithms, other structures may be more appropriate or required in addition to inversion lists for maintaining the required semantic and syntactic information.

Structures used in inverted files: There are several structures that can be used in implementing inverted files: sorted arrays, B-trees, tries, and various hashing structures, or combinations of these structures. The first three of these structures are sorted (lexicographically) indices, and can efficiently support range queries, such as all documents having keywords that start with "comput."

An inverted file implemented as a sorted array structure stores the list of keywords in a sorted array, including the number of documents associated with each keyword and a link to the documents containing that keyword. This array is commonly searched using a standard binary search, although large secondary-storage-based systems will often adapt the array (and its search) to the characteristics of their secondary storage.

The main disadvantage of this approach is that updating the index (for example appending a new keyword) is expensive. On the other hand, sorted arrays are easy to implement and are reasonably fast.

Another implementation structure for an inverted file is a B-tree. More details of B-trees can be found later in this document. Compared with sorted arrays, B-trees use more space. However, updates are much easier and the search time is generally faster, especially if secondary storage is used for the inverted file (instead of memory). The implementation of inverted files using B-trees is more complex than using sorted arrays,

Inverted files can also be implemented using a trie structure. This structure uses the digital decomposition of the set of keywords to represent those keywords. A special trie structure, the Patricia (PAT) tree, is especially useful in information retrieval.

7.2 N-GRAM DATA STRUCTURES

N-Grams are a fixed length consecutive series of "n" characters. N-grams do not care about semantics. Instead they are algorithmically based upon a fixed number of characters. The searchable data structure is transformed into overlapping n-grams, which are then used to create the searchable database. Examples of bigrams, trigrams and pentagrams are given in Figure 7.3 for the word phrase "sea colony." For n-grams, with n greater than two, some systems allow interword symbols to be part of the n-gram set usually excluding the single character with interword symbol option. The symbol # is used to represent the interword symbol which is anyone of a set of symbols (e.g., blank, period, semicolon, colon, etc.). Each of the n-grams created becomes a separate processing tokens and are searchable. It is possible that the same n-gram can be created multiple times from a single word.

se ea co ol lo on ny	Bigrams (no interword symbols)
sea col olo lon ony	Trigrams (no interword symbols)
#se sea ea# #co col olo lon ony ny#	Trigrams (with interword symbol #)
#sea# #colo colon olony lony#	Pentagrams (with interword symbol #)

Figure 7.3: Bigrams, Trigrams and Pentagrams for "sea colony"

As shown in Figure 7.3, an n-gram is a data structure that ignores words and treats the input as a continuous data, optionally limiting its processing by interword symbols. The data structure consists of fixed length overlapping symbol segments that define the searchable processing tokens.

The advantage of n-grams is that they place a finite limit on the number of searchable tokens.

$$MaxSeg_n = (\lambda)^n$$

The maximum number of unique n-grams that can be generated, MaxSeg, can be calculated as a function of n which is the length of the n-grams, and λ which is the number of processable symbols from the alphabet (i.e., non-interword symbols).

Although there is a savings in the number of unique processing tokens and implementation techniques allow for fast processing on minimally sized machines, false hits can occur under some architecture. For example, a system that uses trigrams and does not include interword symbols or the character position of the n-gram in an item finds an item containing "retain detail" when searching for "retail" (i.e., all of the trigrams associated with "retail" are created in the processing of "retain detail"). Inclusion of interword symbols would not have helped in this example. Inclusion of character position of the n-gram would have discovered that the n-grams "ret," "eta," "tai," "ail" that define "retail" are not all consecutively starting within one character of each other. The longer the n-gram, the less likely this type error is to occur because of more information in the word fragment. But the longer the n-gram, the more it provides the same result as full word data structures since most words are included within a single n-gram. Another disadvantage of n-grams is the increased size of inversion lists (or other data structures) that store the linkage data structure. In effect, use of n-grams expands the number of processing tokens by a significant factor. The average word in the English language is between six and seven characters in length. Use of trigrams increases the number of processing tokens by a factor of five (see Figure 7.3) if interword symbols are not included. Thus the inversion lists increase by a factor of five.

Because of the processing token bounds of n-gram data structures, optimized performance techniques can be applied in mapping items to an n-gram searchable structure and in query processing. There is no semantic meaning in a particular n-gram since it is a fragment of processing token and may not represent a concept. Thus n-grams are a poor representation of concepts and their relationships.

7.3 INTRODUCTION TO ADVANCED DATA STRUCTURES AND ALGORITHMS

Advanced data structures support operations on dynamic sets but at a more advanced level. These include data structures such as B Trees, B+ Trees, K D Trees, G trees and R Trees. These allow the user to perform dynamic range of operations such as range search, key search etc. efficiently. We study these data structures in detail, in the rest of the document.

7.4 B TREE

B-trees are balanced search trees designed to work well on magnetic disks or other direct-access secondary storage devices. Many database systems use B-trees, or variants of B-trees, to store information.

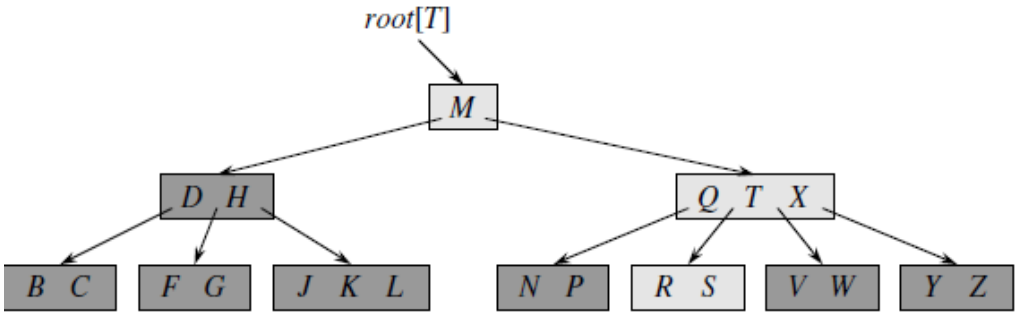


Figure A B-tree whose keys are the consonants of English. An internal node x containing $n[x]$ keys has $n[x] + 1$ children. All leaves are at the same depth in the tree. The lightly shaded nodes are examined in a search for the letter R.

Definition of B-trees: A ***B-tree*** T is a rooted tree (whose root is $root[T]$) having the following properties:

1. Every node x has the following fields:
   a) n[x], the number of keys currently stored in node x,
   b) the n[x] keys themselves, stored in nondecreasing order, so that $key_1[x] \le key_2[x] \le \cdots \le key_{n[x]}[x]$,
   c) leaf[x], a boolean value that is TRUE if x is a leaf and FALSE if x is an internal node.
2. Each internal node x also contains n[x]+1 pointers $c_1[x], c_2[x], \ldots, c_{n[x]+1}[x]$ to its children. Leaf nodes have no children, so their ci fields are undefined.
3. The keys $key_i[x]$ separate the ranges of keys stored in each subtree: if k_i is any key stored in the subtree with root $c_i[x]$, then
   $$k_1 \le key_1[x] \le k_2 \le key_2[x] \le \cdots \le key_{n[x]}[x] \le k_{n[x]+1} .$$
4. All leaves have the same depth, which is the tree's height h.
5. There are lower and upper bounds on the number of keys a node can contain. These bounds can be expressed in terms of a fixed integer $t \ge 2$ called the ***minimum*** degree of the B-tree:
   a. Every node other than the root must have at least $t - 1$ keys. Every internal node other than the root thus has at least t children. If the tree is nonempty, the root must have at least one key.

b. Every node can contain at most 2t − 1 keys. Therefore, an internal node can have at most 2t children. We say that a node is full if it contains exactly 2t − 1 keys.1

The simplest B-tree occurs when t = 2. Every internal node then has either 2, 3, or 4 children, and we have a 2-3-4 tree. In practice, however, much larger values of t are typically used.

The height of a B-tree: The number of disk accesses required for most operations on a B-tree is proportional to the height of the B-tree. If $n \geq 1$, then for any n-key B-tree T of height h and minimum degree $t \geq 2$, $h \leq \log_t[(n + 1)/2]$

Searching a B-tree: Searching a B-tree is much like searching a binary search tree, except that instead of making a binary, or "two-way," branching decision at each node, we make a multiway branching decision according to the number of the node's children. More precisely, at each internal node x, we make an (n[x]+1)-way branching decision. SEARCH_BTREE procedure takes as input a pointer to the root node x of a subtree and a key k to be searched for in that subtree. The top-level call is thus of the form SEARCH_BTREE(root[T], k). If k is in the B-tree, SEARCH_BTREE returns the ordered pair (y, i) consisting of a node y and an index i such that key_i [y] = k. Otherwise, the value NIL is returned.

Procedure: **SEARCH_BTREE** *(x, k)*

```
1 i ← 1
2 while i ≤ n[x] and k > key_i [x]
3       do i ←i + 1
4       if i ≤ n[x] and k = key_i [x]
5             then return (x, i )
6       if leaf [x]
7             then return NIL
8             else Process(c_i[x])
9             return SEARCH_BTREE(c_i[x], k)
```

Inserting a key into a B-tree

Inserting a key into a B-tree is significantly more complicated than inserting a key into a binary search tree. As with binary search trees, we search for the leaf position at which to

insert the new key. With a B-tree, however, we cannot simply create a new leaf node and insert it, as the resulting tree would fail to be a valid B-tree. Instead, we insert the new key into an existing leaf node. Since we cannot insert a key into a leaf node that is full, we introduce an operation that splits a full node y (having 2t − 1 keys) around its median key $key_t[y]$ into two nodes having t – 1 keys each. The median key moves up into y's parent to identify the dividing point between the two new trees. But if y's parent is also full, it must be split before the new key can be inserted, and thus this need to split full nodes can propagate all the way up the tree.

Splitting a node in a B-tree: The procedure B-TREE-SPLIT-CHILD takes as input a nonfull internal node x (assumed to be in main memory), an index i , and a node y (also assumed to be in main memory) such that y = c_i [x] is a full child of x. The procedure then splits this child in two and adjusts x so that it has an additional child. (To split a full root, we will first make the root a child of a new empty root node, so that we can use B-TREE-SPLIT-CHILD. The tree thus grows in height by one; splitting is the only means by which the tree grows.)

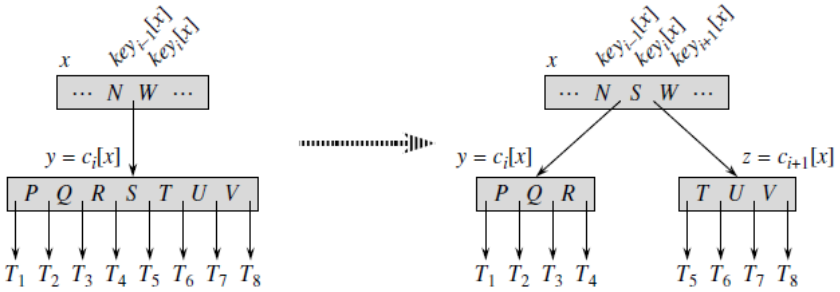


Figure 7.4: Splitting a node with t = 4. Node y is split into two nodes, y and z, and the median key S of y is moved up into y's parent.

Procedure: **B-TREE-SPLIT-CHILD**(x, i, y)

1 z ← ALLOCATE-NODE()
2 leaf [z]← leaf [y]
3 n[z]← t − 1
4 for j ← 1 to t − 1
5 do key_j [z]← key_{j+t} [y]
6 if not leaf [y]
7 then for j ← 1 to t
8 do c_j [z]← c_{j+t} [y]
9 n[y]← t − 1
10 for j ← n[x] + 1 down to i + 1
11 do c_{j+1}[x] ← cj [x]
12 c_{i+1}[x] ← z

```
13 for j ← n[x] downto i
14       do key j+1[x] ← key j [x]
15 keyi [x] ← keyt [y]
16 n[x] ← n[x] + 1
17 DISK-WRITE(y)
18 DISK-WRITE(z)
19 DISK-WRITE(x)
```

Inserting a key into a B-tree in a single pass down the tree

We insert a key k into a B-tree T of height h in a single pass down the tree, requiring O(h) disk accesses. The CPU time required is $O(th) = O(t \log_t n)$. The B-TREE-INSERT procedure uses B-TREE-SPLIT-CHILD to guarantee that the recursion never descends to a full node.

Procedure: **B-TREE-INSERT**(T, k)

```
1 r ← root[T ]
2 if n[r] = 2t − 1
3       then s ← ALLOCATE-NODE()
4                root[T ]← s
5                leaf [s] ← FALSE
6                n[s] ← 0
7                c1[s] ←r
8                B-TREE-SPLIT-CHILD(s, 1, r)
9                B-TREE-INSERT-NONFULL(s, k)
10      else B-TREE-INSERT-NONFULL(r, k)
```

Lines 3–9 handle the case in which the root node r is full: the root is split and a new node s (having two children) becomes the root. Splitting the root is the only way to increase the height of a B-tree. Figure 18.6 illustrates this case. Unlike a binary search tree, a B-tree increases in height at the top instead of at the bottom. The procedure finishes by calling B-TREE-INSERT-NONFULL to perform the insertion of key k in the tree rooted at the nonfull root node. B-TREE-INSERT-NONFULL recurses as necessary down the tree, at all times guaranteeing that the node to which it recurses is not full by calling B-TREE-SPLIT-CHILD as necessary.

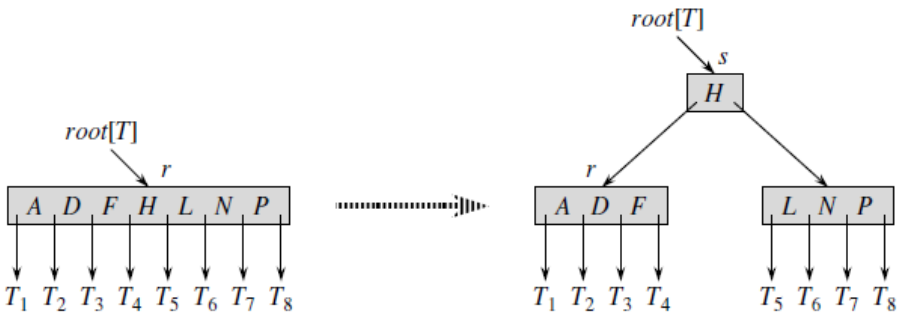


Figure 7.5: Splitting the root with t = 4. Root node r is split in two, and a new root node s is created. The new root contains the median key of r and has the two halves of r as children. The B-tree grows in height by one when the root is split.

The auxiliary recursive procedure B-TREE-INSERT-NONFULL inserts key k into node x, which is assumed to be nonfull when the procedure is called. The operation of B-TREE-INSERT and the recursive operation of B-TREE-INSERT-NONFULL guarantee that this assumption is true.

Procedure: **B-TREE-INSERT-NONFULL**(x, k)

```
1 i ← n[x]
2 if leaf [x]
3       then while i ≥ 1 and k < keyi [x]
4                 do keyi+1[x] ← keyi [x]
5                 i ←i − 1
6       keyi+1[x] ← k
7       n[x] ← n[x] + 1
8       DISK-WRITE(x)
9 else while i ≥ 1 and k < keyi [x]
10                do i ←i − 1
11      i ←i + 1
12      DISK-READ(ci [x])
13      if n[ci [x]] = 2t − 1
14                then B-TREE-SPLIT-CHILD(x, i, ci [x])
15                     if k > keyi [x]
16                        then i ←i + 1
17      B-TREE-INSERT-NONFULL(ci [x], k)
```

The B-TREE-INSERT-NONFULL procedure works as follows. Lines 3–8 handle the case in which x is a leaf node by inserting key k into x. If x is not a leaf node, then we must insert k into the appropriate leaf node in the subtree rooted at internal node x. In this case, lines 9–11 determine the child of x to which the recursion descends. Line 13 detects whether the recursion would descend to a full child, in which case line 14 uses B-TREE-SPLIT-CHILD to split that child into two nonfull children, and lines 15–16 determine which of the two children is now the correct one to descend to. The net effect of lines 13–16 is thus to guarantee that the procedure never recurses to a full node. Line 17 then recurses to insert k into the appropriate subtree. Figure 18.7 illustrates the various cases of inserting into a B-tree. The number of disk accesses performed by B-TREE-INSERT is O(h) for a Btree of height h, since only O(1) DISK-READ and DISK-WRITE operations are performed between calls to B-TREE-INSERT-NONFULL. The total CPU time used is O(th) = O(t logt n). Since B-

TREE-INSERT-NONFULL is tail-recursive, it can be alternatively implemented as a while loop, demonstrating that the number of pages that need to be in main memory at any time is O(1).

Deleting a key from a B-tree: Deletion from a B-tree is analogous to insertion but a little more complicated, because a key may be deleted from any node—not just a leaf—and deletion from an internal node requires that the node's children be rearranged. As in insertion, we must guard against deletion producing a tree whose structure violates the B-tree properties. Just as we had to ensure that a node didn't get too big due to insertion, we must ensure that a node doesn't get too small during deletion (except that the root is allowed to have fewer than the minimum number $t - 1$ of keys, though it is not allowed to have more than the maximum number $2t - 1$ of keys). Just as a simple insertion algorithm might have to back up if a node on the path to where the key was to be inserted was full, a simple approach to deletion might have to back up if a node (other than the root) along the path to where the key is to be deleted has the minimum number of keys.

7.5 B+ TREE

B+ Tree is a B-tree in which keys are stored in the leaves. The main difference is that nodes of a B+-tree will point to many children nodes rather than being limited to only two. Since goal is to minimize disk accesses whenever we are trying to locate records, the height of the multiway search tree as small as possible. This goal is achieved by having the tree branch in large amounts at each node.

A B+ tree of order m is a tree where each internal node contains up to m branches (children nodes) and thus store up to m-1 search key values in a binary search tree, only one key value is needed since there are just two children nodes that an internal node can have. m is also known as the branching factor or the fanout of the tree.

1. The B+-tree stores records (or pointers to actual records) only at the leaf nodes, which are all found at the same level in the tree, so the tree is always height balanced.
2. All internal nodes, except the root, have between Ceiling(m/2) and m children
3. The root is either a leaf or has at least two children.
4. Internal nodes store search key values, and are used only as placeholders to guide the search. The number of search key values in each internal node is one less than the number of its non-empty children, and these keys partition the keys in the children in

the fashion of a search tree. The keys are stored in non-decreasing order (i.e. sorted in lexicographical order).

5. Depending on the size of a record as compared to the size of a key, a leaf node in a B+-tree of order m may store more or less than m records. Typically this is based on the size of a disk block, the size of a record pointer, etcetera. The leaf pages must store enough records to remain at least half full.
6. The leaf nodes of a B+-tree are linked together to form a linked list. This is done so that the records can be retrieved sequentially without accessing the B+-tree index. This also supports fast processing of range-search queries as will be described later.

Searching for records that satisfy a simple condition: To understand the B+-tree operations more clearly, assume, without loss of generality, that there is a table whose primary is a single attribute and that it has a B+-tree index organized on the PK attribute of the table. To retrieve records, queries are written with conditions that describe the values that the desired records are to have. The most basic search on a table to retrieve a single record given its PK value K. Search in a B+-tree is an alternating two-step process, beginning with the root node of the B+-tree. Say that the search is for the record with key value K -there can only be one record because we assume that the index is built on the PK attribute of the table.

1. Perform a binary search on the search key values in the current node -- recall that the search key values in a node are sorted and that the search starts with the root of the tree. We want to find the key Ki such that $Ki \leq K < Ki+1$.
2. If the current node is an internal node, follow the proper branch associated with the key Ki by loading the disk page corresponding to the node and repeat the search process at that node.
3. If the current node is a leaf, then:
    a. If K=Ki, then the record exists in the table and we can return the record associated with Ki
    b. Otherwise, K is not found among the search key values at the leaf, we report that there is no record in the table with the value K.

Inserting into a B+-tree: Insertion in a B+-tree is similar to inserting into other search trees, a new record is always inserted at one of the leaf nodes. The complexity added is that insertion could overflow a leaf node that is already full. When such overflow situations occur

a brand new leaf node is added to the B+-tree at the same level as the other leaf nodes. The steps to insert into a B+-tree are:

1. Follow the path that is traversed as if a Search is being performed on the key of the new record to be inserted.
2. The leaf page L that is reached is the node where the new record is to be indexed.
3. If L is not full then an index entry is created that includes the seach key value of the new row and a reference to where new row is in the data file. We are done; this is the easy case!
4. If L is full, then a new leaf node L_{new} is introduced to the B+-tree as a right sibling of L. The keys in L along with the an index entry for the new record are distributed evenly among L and L_{new}. Lnew is inserted in the linked list of leaf nodes just to the right of L. We must now link L_{new} to the tree and since L_{new} is to be a sibling of L, it will then be pointed to by the partent of L. The smallest key value of L_{new} is copied and inserted into the parent of L -- which will also be the parent of L_{new}. This entire step is known as commonly referred to as a split of a leaf node.
    a) If the parent P of L is full, then it is split in turn. However, this **split of an internal node** is a bit different. The search key values of P and the new inserted key must still be distributed evenly among P and the new page introduced as a sibling of P. In this split, however, the **middle key is moved to the node above** -- note, that unlike splitting a leaf node where the middle key is copied and inserted into the parent, when you split an internal node the middle key is removed from the node being split and inserted into the parent node. This splitting of nodes may continue upwards on the tree.
    b) When a key is added to a full root, then the root splits into two and the middle key is promoted to become the new root. This is the only way for a B+-tree to increase in height -- when split cascades the entire height of the tree from the leaf to the root.

Deletion: Deletion from a B+-tree again needs to be sure to maintain the property that all nodes must be at least half full. The complexity added is that deletion could underflow a leaf node that has only the minimum number of entries allowed. When such underflow situations take place, adjacent sibling nodes are examined; if one of them has more than the minimum entries required then some of its entries are taken from it to prevent a node from

underflowing. Otherwise, if both adjacent sibling nodes are also at their minimum, then two of these nodes are merged into a single node. The steps to delete from a B+-tree are:

1. Perform the search process on the key of the record to be deleted. This search will end at a leaf L.
2. If the leaf L contains more than the minimum number of elements (more than m/2 - 1), then the index entry for the record to be removed can be safely deleted from the leaf with no further action.
3. If the leaf contains the minimum number of entries, then the deleted entry is replaced with another entry that can take its place while maintaining the correct order. To find such entries, we inspect the two sibling leaf nodes L_{left} and Lright adjacent to L -- at most one of these may not exist.
    a) If one of these leaf nodes has more than the minimum number of entries, then enough records are transferred from this sibling so that both nodes have the same number of records. This is a heuristic and is done to delay a future underflow as long as possible; otherwise, only one entry need be transferred. The placeholder key value of the parent node may need to be revised.
    b) If both L_{left} and L_{right} have only the minimum number of entries, then L gives its records to one of its siblings and it is removed from the tree. The new leaf will contain no more than the maximum number of entries allowed. This merge process combines two subtrees of the parent, so the separating entry at the parent needs to be removed -- this may in turn cause the parent node to underflow; such an underflow is handled the same way that an underflow of a leaf node.
    c) If the last two children of the root merge together into one node, then this merged node becomes the new root and the tree loses a level.

Most database systems use Indexes built on some form of a B+-tree due to its many advantages, in particular its support for range queries.

7.6 KD TREE

Introduction to Range Searching: Databases store records in a multidimensional space and the queries about records are transformed into the queries over this set of points. Every point in the space will have some information of person associated with it.

Consider the example of the database of personal administration where the general information of each employee is stored. Consider an example of query where we want to report all employees born between 1950 and 1955, who earns between Rs.3000 and Rs.4000 per month. The query will report all the points that whose frost co-ordinate lies between 1950 and 1955, and second co-ordinate lies between 3000 and 4000.

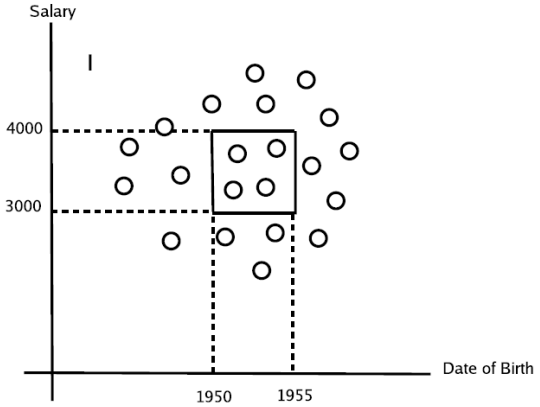


Figure 7.6: interpreting a database query geometrically

In general if we are interested in answering queries on k-fields of the records in our database, we transforms the records to points in k-dimensional space. Such a query is called rectangular range query, or an orthogonal range query. Range search consist either of retrieving (report problems) or of counting (count problems) all points contained within the query domain.

1-Dimensional Range Searching: Let us consider the set of points P=p1,p2.....pn. We have to search the range [x,x'] and we have to report which points lies in that range. To solve the problem of range searching we use the data structure known as *balanced binary search tree* T. The leaves of the T store the points of P and the internal nodes of T will store the splitting values to guide the search. Let x_v denote the value stored at each split node v. The left subtree of the node v contains all points smaller than or equal to x_v, and the right subtree contains all the points strictly greater than x_v.

Let we search with x and x' in T. μ and μ' be the two leaves where the searches end, respectively. Then the points in the interval [x:x'] are stored in the leaves in between μ and μ' including μ and μ'. To find the leaves between μ and μ', we select the tree rooted at nodes v

in between the two search paths whose parent are on the search path. To find these nodes we first find the node v_{split} where the paths to x and x' splits.

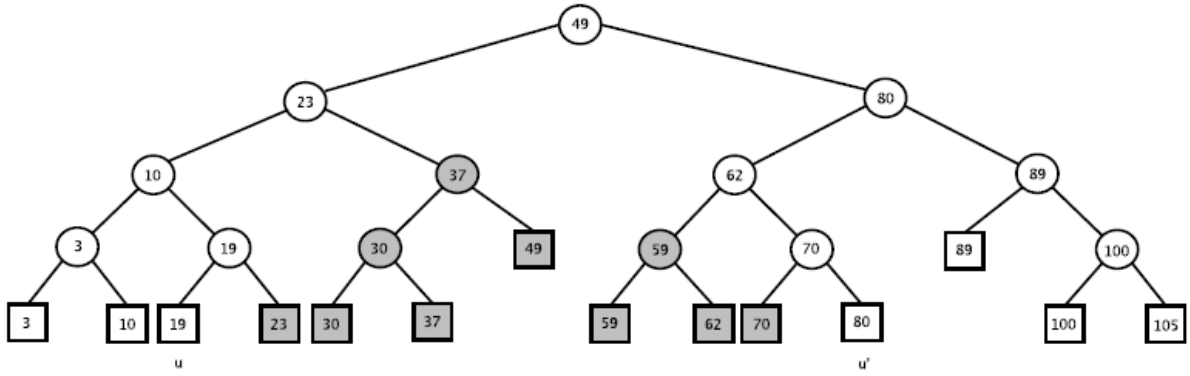


Figure 7.7: 1D range query in a binary search tree

Algorithm: To find the split node.

Let us consider ***lc(v)*** and ***rc(v)*** denote the left and right child, respectively, of the node v.

Procedure name: **FINDSPLITNODE**

Input: A Tree T and two values x and x' with $x \leq x'$.

Output: The node v where the paths to x and x' split, or the leaf where both path ends.

1: $v \leftarrow$ root(T).

2: **while** v is not a leaf and ($x' \leq x_v$ or $x > x_v$)

3: do if $x' \leq x_v$

4: then $v \leftarrow lc(v)$

5: else $v \leftarrow rc(v)$

6: return v

Starting from v_{split} we then follow the search path of x. At each node where the path goes left, we report all the leaves in the right subtree, because this subtree is in between the the two search paths. Similarly, we follow the path of x' and we report the leaves in the left subtree of the node where the path goes right. Finally we check the points stored at the leaves whether they lies in the range [x , x'] or not.

Algorithm: To perform 1D range query

Now we see the query algorithm 1DRANGEQUERY which assumes the subroutine REPORTSUBTREE, which traverses the subtree rooted at the node and reports all the stored

at its leaves. This subroutine takes the amount of time linear in the number of reported points; this is because the number of internal nodes of any binary tree is less than its internal node.

Procedure name: **1DRANGEQUERY**(T,[X:X'])
Input: A binary search tree T and a range [x,x'].
Output: All points stored in T that lie in the range.

1: v_{split} ← FINDSPLITNODE(T,x,x').
2: if v_{split} is a leaf
3: then check if the point stored at v_{split} must be reported.
4: else (* Follow the path to x and report the points in subtree right of the path*).
5: v ←lc(vsplit)
6: while v is not a leaf.
7: do if x' _ xv
8: then REPORTSUBTREE(rc(v))
9: v ← lc(v)
10: else v rc(v)
11: check if the point stored at the leaf v must be reported.
12: Similarly, follow the path to x', reports the points subtree left of the path, and check if the point stored at the leaf where the path ends must be reported.

Complexity analysis: The data structure binary search tree uses O(n) storage and it can be built in O(n logn) time. In worst case all the points could be in query range, so the query time will be Θ(n). The query time of Θ (n) cannot be avoided when we have to report all the points. Therefore we shall give more refined analysis of query time. The refined analysis takes not only n, the number of points in the set P, into account but also k, the number of reported points.

As we know that the REPORTSUBTREE is linear in the number of reported points, then the total time spent in all such calls is O(k). The remaining nodes that are visited are the nodes of the search path of x and x'. Because T is balanced, these paths have a length O(log n). The time we spent at each node is O(1), so the total time spent in these nodes is O(log n), which gives a query time of O(logn + k).

Kd – Trees: Consider the 2-dimensional rectangular range searching problem. Let P be the set of n points in the plane. The basic assumption is no two point have same x-coordinate, and no two points have same y-coordinate.

Let's consider the following recursive definition of the binary search tree : the set of (1-dimensional) points is split into two subsets of roughly equal size, one subset contains the point smaller than or equal to splitting value, the other contains the points larger than splitting value. The splitting value is stored at the root and the two subsets are stored recursively in two subtrees.

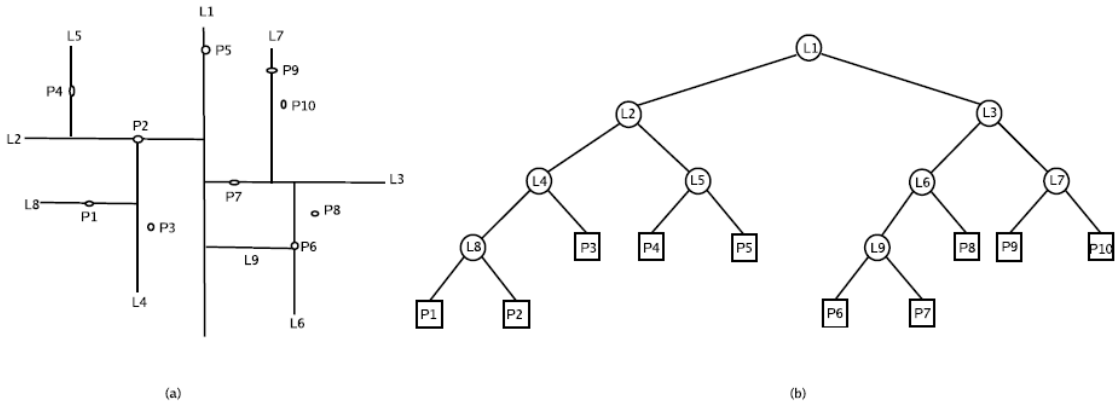


Figure 7.8: A Kd-tree: (a) The way the plane is devided. (b) Corresponding binary tree

Each point has its x-coordinate and y-coordinate. Therefore we first split on x-coordinate and then on y-coordinate, then again on x-coordinate, and so on. At the root we split the set P with vertical line l into two subsets of roughly equal size. This is done by finding the median x- coordinate of the points and drawing the vertical line through it. The splitting line is stored at the root. P_{left}, the subset of points to left is stored in the left subtree, and P_{right}, the subset of points to right is stored in the right subtree. At the left child of the root we split the P_{left} into two subsets with a horizontal line. This is done by finding the median y-coordinate if the points in P_{left}. The points below or on it are stored in the left subtree, and the points above are stored in right subtree. The left child itself store the splitting line. Similarly P_{right} is split with a horizontal line, which are stored in the left and right subtree of the right child. At the grandchildren of the root, we split again with a vertical line. In general, we split with a vertical line at nodes whose depth is even, and we split with horizontal line whose depth is odd.

Algorithm: Building Kd tree: Let us consider the procedure for constructing the kd-tree. It has two parameters, a set of points and an integer. The first parameter is set for which we

want to build kd-tree, initially this the set P. The second parameter is the depth of the root of the subtree that the recursive call constructs. Initially the depth parameter is zero. The procedure returns the root of the kd-tree.

Procedure name **BUILDKDTREE**(P,depth)
Input: A set of points P and the current depth depth.
Output: The root of the kd-tree storing P.
1: if P contains only one point
2: then return a leaf storing this point
3: else if depth is even
4: then Split P into two subsets with a vertical line l through the median x-coordinate of the points in P. Let P1 be the set of points to the left of l or on l, and let P2 be the set of points to the right of l.
5: else Split P into two subsets with a horizontal line l through the median y-coordinate of the points in P.Let P1 be the set of points to the below of l or on l, and let P2 be the set of points above l.
6: $v_{left} \leftarrow$ BUILDKDTREE(P1, depth +1).
7: $v_{right} \leftarrow$ BUILDKDTREE(P2, depth +1).
8: Create a node v storing l, make v_{left} the left child of v, and make v_{right} the right child of v.
9: return v.

Construction time of 2-dimensional kd-tree

The most expensive step is to find the median. The median can be find in linear time, but linear time median finding algorithms are rather complicated. So first presort the set of points on x-coordinate and then on y-coordinate. The parameter set P is now passed to the procedure in the form of two sorted list. Given the two sorted list it is easy to find the median in linear time. Hence the building time satisfies the recurrence,

$T(n)=O(1)$, if $n=1$,
$O(n)+2T(dn/2e)$,if $n>1$

which solves to O(n log n).

Because the kd-tree is the binary tree, and every leaf and internal node uses O(1) storage, therefore the total storage is O(n).

When a region is fully contained in the query rectangle then report all the points stored at its subtree. When traverse reaches a leaf, we check whether the point is contained in the query region and, if so, report it. Let us consider the diagram given below.

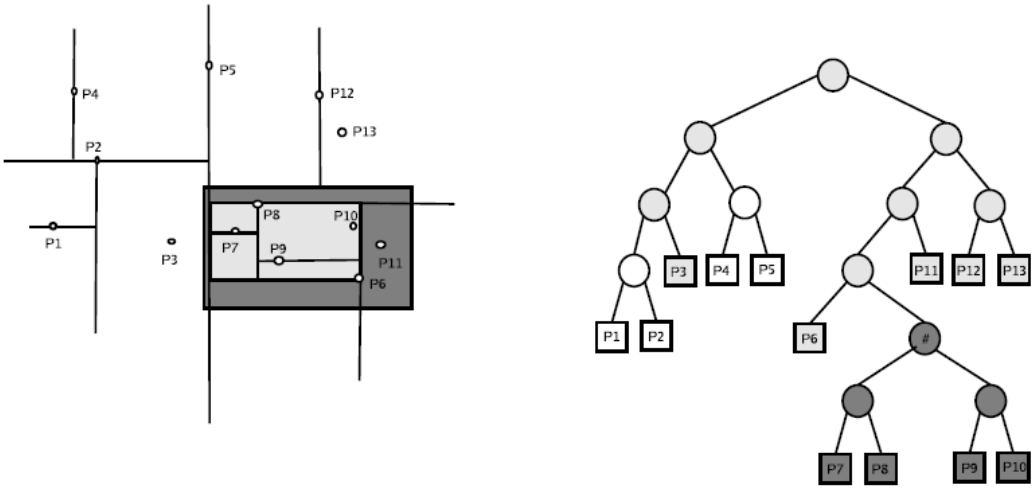


Figure 7.9: A query in a kdree

The grey nodes are visited when we query with grey rectangle. The node marked with star corresponds to a region that is completely contained in the query rectangle, which is shown by darker rectangle in figure. Hence, the dark grey subtree rooted at this node is traversed and all points stored in it are reported. The other leaves that are visited corresponds to region that are only partially inside the query rectangle. Hence, the points stored in them must be tested for inclusion in the query range; this results in point P6 and P11 being reported, and points P3, P12 and P13 not being reported.

Algorithm: Searching kd-tree

Let us consider the procedure for searching the kd-tree. It has two parameters, the root of the kd-tree and the query range R.

Procedure name: **SEARCHKDTREE**(v,R)

Input: The root of (a subtree of) a kd-tree, and a range R.

Output: All points at leaves below v that lies in range.

1: if v is a leaf

2: then Report the stored at v if it lies in R

3: else if region(lv(c)) is fully contained in R

4: then REPORTSUBTREE(lc(v))

5: else if region(lc(v)) intersects R

6: then SEARCHKDTREE(lc(v),R)

7: if region(rv(c)) is fully contained in R

8: then REPORTSUBTREE(rc(v))

9: else if region(lc(v)) intersects R

10: then SEARCHKDTREE(lc(v),R)

The region corresponding to the left child of a node v at even depth can be computed from region(v) as follows :

$$region(lc(v)) = region(v) \cap l(v)^{left}$$

where l(v) is the splitting line stored at v, and $l(v)^{left}$ is the half plane to the left of and including l(v).

Complexity analysis

Let l be the vertical line, and T be a kd-tree. Let l(root(T)) be the splitting line stored at the root of the kd-tree. The line l intersects either the region to the left of l(root(T)) or the region to the right of l(root(T)), but not both. This observation seems to imply that Q(n), the number of intersected regions in the kd-tree storing a set of n points, satisfies the recurrence Q(n)=1+Q(n/2). But this is not true because the splitting lines are horizontal at the children of the root. This means that if the line l intersects the region(lc(root(T))), then it will always intersect the regions corresponding to both children of lc(root(T)). Hence the recurrence we get is incorrect. To write the correct recurrence for Q(n) we go down two steps in tree. Each of the four nodes at depth two in the tree corresponds to a region containing n/4 points. Two of the four nodes correspond to intersected regions, so we have to count the number of intersected regions in these subtrees recursively. Moreover, l intersects the regions of the root and of the root and of one of its children. Hence, Q(n) satisfies the recurrence

$$Q(n)=O(1),\text{if } n=1,$$
$$2+2Q(n/4),\text{if } n>1.$$

This recurrence solves to $Q(n)=O(\sqrt{n})$. In other words, any vertical line intersects O(pn) regions in kd-tree.Similarly, horizontal line intersects $O(\sqrt{n})$regions. The total number of regions intersected by the boundary of a rectangular range query is bounded by $O(\sqrt{n})$.

A kd-tree for a set P of n points can be build in O(n logn) time. A rectangular range query on the kd-tree takes $O(\sqrt{n}+ k)$ time, where k is the number of reported points.

Kd-trees can be also be used for higher-dimensional spaces. The construction algorithm is very similar to the planar case. At the root, we split the set of points into two subsets of roughly the same size by a hyperplane perpendicular to the x1-axis. In other words, at the root the point set is partitioned based on the _rst coordinate of the points. At the children of the root the partition is based on second coordinate, at nodes at depth two on the third coordinate, and so on, until depth of d-1 we partition on last coordinate. Ai depth d we start all over again, partitioning on _rst coordinate. The recursion stops only when one point is left, which is then stored at the leaf. Because a d-dimensional kd-tree for a set of n points is a binary tree with n leaves, it uses O(n) storage. The construction time is O(n log n).

Nodes in a d-dimensional kd-tree corresponds to regions, as in the plane. The query algorithm is its those nodes whose regions are properly intersected by the query range, and traverses subtree that are rooted at nodes whose region is fully contained in the query range. It can be hown that the query time is bounded by $O(n^{1-1/d}+k)$.

7.7 G TREE

Introduction: G-tree (or grid tree) is used for organizing multidimensional data. The data structure combines the features of grids and B-trees in a novel manner. It also exploits an ordering property that numbers the partitions in such a way that partitions that are spatially close to one another in a multidimensional space are also close in terms of their partition numbers. This structure adapts well to dynamic data spaces with a high frequency of insertions and deletions, and to nonuniform distributions of data.

Data Structure: We assume that each data point (or tuple) lies in an n-dimensional space, and the dimensions are numbered 1, 2 , . . . , n . It is also assumed that the range within which the points lie along dimension i is $[l_i, h_i]$.

A multidimensional data space is divided into several partitions, each containing not more than a maximum number of entries, denoted by the parameter ***max-entries***. Each partition corresponds to one disk page or bucket and, upon becoming full, is split into two. The main features of scheme are:

- The data space is divided into non-overlapping partitions of variable size.
- Each partition is assigned a unique partition number.
- A total ordering is defined on the partition numbers, and they are stored in a B-tree.
- Empty partitions are not stored in the tree to save space.

Partition Numbering and Splitting: A partition is numbered as a binary string of 0's and 1's. Successive parts of Fig. 7.10 show how partitions are split and new partitions created. In this example, there are only two dimensions, 1 and 2 (or dimensions X and Y, respectively). Initially, the entire data space is divided into two partitions by splitting its range along dimension 1 (or X axis) into two equal sub-partitions, numbered 0 and 1 (see Fig. 7.10(a)). As more points are added to a partition and it becomes full, it is subdivided to create two new partitions of equal size. (In Fig. 7.10, we assume that max-entries is 2, i.e., each partition can accommodate only two entries.)

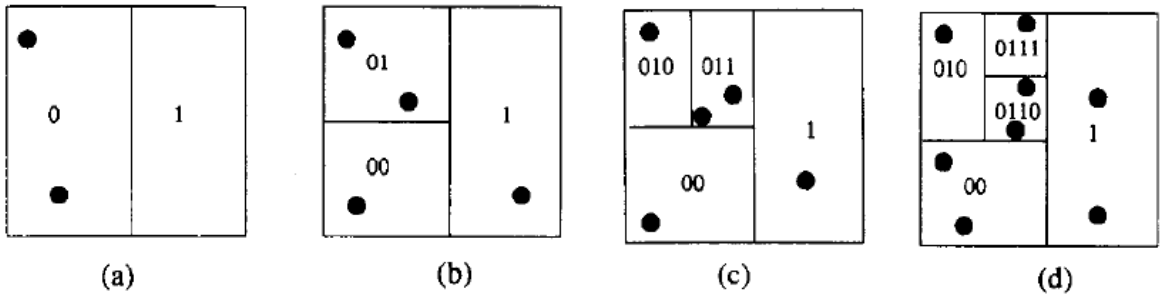


Figure 7.10: Partitioning scheme.

With only two dimensions, the splitting dimension alternates; in general, with n dimensions, the splitting dimension recycles with a periodicity of n such that each dimension appears once in a cycle.

Partition Number Arithmetic: The nonempty partitions that span the data space are maintained in a B-tree-like structure, called the Gtree. A leaf-level page in the G-tree points to a disk page containing all the points that lie in a partition, while higher level pages point to pages at the next lower level. In this subsection, we first define various operations on partitions (such as <, >, etc.), and then show that the partition numbers in a G-tree are totally ordered.

Insertion Algorithm: The first step in inserting a point is to compute its partition number. Since it is not possible to exactly determine the partition to which a point belongs, an approximate, initial partition number is computed by assuming that the point will be inserted into a partition with dimensions equal to those of the smallest partition created so far. The smallest partition is the one with the most number of bits in its number. The function assign is called with the coordinates of the point (x_1, x_2, ..., x_n,) and the number of bits b in the desired

partition number as parameters (assuming that the number of bits in the smallest partition is b). It returns an initial partition number P_{in} which is b bits long and contains the point.

The next step is to use the initial partition number P_{in} as a handle to search the G-tree and return P_{act}, the actual partition to which the new point belongs. This is performed by function search. The G-tree is searched for a partition P' such that $P_{in} \subseteq P'$. If such a P' is found, the point belongs to this partition, and Pact is set equal to P'. On the other hand, if partition P' is not found, the search terminates as soon as either the last partition in the G-tree is encountered, or the first partition number higher than P_{in} is located. In both cases, this means that a new partition must be inserted. This new partition is Pin itself or its largest ancestor that does not overlap with an existing partition in the G-tree. In order to determine the largest ancestor, successive ancestors of P_{in} must be compared with P_{prev} and P_{next} (the next lower and the next higher entries than P_{in} in the G-tree, respectively). This is performed within the while loop in function ***search***.

Once the correct partition number is found, a check is performed to see if this partition can accommodate one more entry. If so, the new entry is added to this partition; otherwise, the partition must be split. This means that two new child partitions of Pact must be created by appending a 0 and a 1 to it. The two new partitions created by splitting Pact are denoted P0 and P1. Partition P_{act} is deleted from the G-tree, and all the points that belong to it are reallocated to P0 and P1. The two new partitions, if nonempty, are inserted into the G-tree. On the other hand, if a partition is empty, then our algorithm does not require that it should be inserted into the G-tree. This helps in reducing the size of the tree.

Unless all the points go into only one of P0 or P1, the split will result in the creation of space for an additional point in both P0 and P1. In this case, function assign is called again to determine the partition to which the point belongs, and the point is inserted into the partition. On the other hand, if all the points upon splitting go into only one of P0 or P1, then clearly that partition would still remain full. This means that if the newly inserted point also belongs to this partition, which is already full, then another split is necessary. The splitting must be repeated until the total number of points in P_{act} are distributed across more than one partition, and space is made for the new point.

Procedure: **Insert**($x_1, x_2, \ldots, x_n$) /*coordinates of point to be inserted */
{
    P_{in} = assign($x_1, x_2, \ldots, x_n$)
    P_{act} = search(P_{in});

```
        flag = 0;
        while (flag ≠1)
        if (num-points(P_act)< max-entries)
        {
                insert(x_1, x_2, . . , x_n ) into partition P_act;        /*insert point into partition P_act */
                flag = 1;
        }
        else
        {        /*split a partition*/
                P0 = concatenate(P_act, '0');/*P0 and P1 are child partitions of P_act*/
                P1 = concatenate(P_act, '1');
                delete P_act from G-tree;
                reallocate all points in P_act to P0 and P1;
                if (num_points(P0) > 0) /*if P0 is a nonempty partition*/
                        insert P0 into G-tree;
                if (num_points(P1) > 0) /*if P1 is a nonempty partition*/
                        insert P1 into G-tree;
                P_in = assign (x_1, x_2, . . , x_n, size(P0));
                P_act = search(P;,);
        }
}

Procedure: assign(x_1, x_2, . . , x_n) /*computes an initial partition number */
{
        P="". /* null string */
        for (k = 1; k ≤ b; k + +) /* repeat b times */
        {
                i = k mod n;
                if (x < (li+hi)/2)
                {
                        concatenate "0" to P;
                        hi = (li+hi)/2
                }
                else
                {
                        concatenate "1" to P;
                        li = (li+hi)/2
                }
        }

        return (P)
}


Procedure:  search(P_in) /* search G-tree and find exact partition to which point belongs, given an initial partition P,,*/
{
        P = smallest partition number in G-tree;
        while ( P < P_in)
                P=P_next          /* search for P_in in G-tree */
```

```
        P1 = P;
        If(Pin ⊆ P')      /* partition to which point belongs exists */
                Pact =P';
        Else    /* partition to which point belongs does not exist */
        {
                Pact= Pin          /*find the new partition to insert */
                while (Pprev < parent(Pact) < Pnext) /*find largest missing partition */
                        Pact = parent(Pact);
                insert Pact into G-tree;
        }
        retum(Pact );
}
```

Deletion Algorithm: In deleting a point, the first step is to determine the partition in which it lies. As in the case of insertions, an approximate, initial partition P_{in} is computed by function assign. Next, a search is carried out in the G-tree for an actual partition P_{act} such that $P_{act} \supseteq P_{in}$. If such a P_{act} is found, the given point is searched in P_{act} and deleted from it. On the other hand, if no P_{act} such that $P_{act} \supseteq P_{in}$, is found, it means that the given point does not exist.

Once partition P_{act} is located, the next step is to check if P_{act} and compl(P_{act}) (if one exists) can be merged into a single partition. If the total number of points in partitions P_{act} and compl(P_{act}) together is less than or equal to max-entries, then it is possible to merge the two partitions and replace them by their parent partition. In this case, all the points belonging to Pact and compl(P_{act}) are assigned to parent(P_{act}), partitions P_{act} and compl(P_{act}) are deleted from the G-tree and parent(Pact) is inserted into it.

```
Procedure:  Delete(x1, x2, . . , xn) /*coordinates of point to be deleted */
{
        Pin = assign(x1, x2, . . , xn,b)
        P = smallest partition in G-tree;
        while (Pin< P)
                P = Pnext          /*search for  Pin in G-tree */
        If(Pin ⊆ P) /*partition containing the point exists in the tree*/
        {
                Pact = P;
                delete point (x1, x2, . . , xn) from Pact;
                while (numqoints(Pa,t) + num_points(compl(Pact)) ≤ max-entries )
                {
                        reassign points in Pact and compl(Pact) to parent(Pact);
                        delete Pact and compl(Pact) from G-tree;
                        insert parent(Pact) into G-tree;
                        Pact = parent(Pact);
                }
```

```
        }
        Else    /*partition containing the point does not exist */
        {
                print("error - point does not exist");
        }
}
```

Example: Fig. 7.11 shows the partitioning of a two-dimensional space resulting from the addition and deletion of several points. Each dot represents a data point, and it is assumed that the maximum number of points in a partition is two. Since the data distribution is nonuniform, the partition sizes vary considerably.

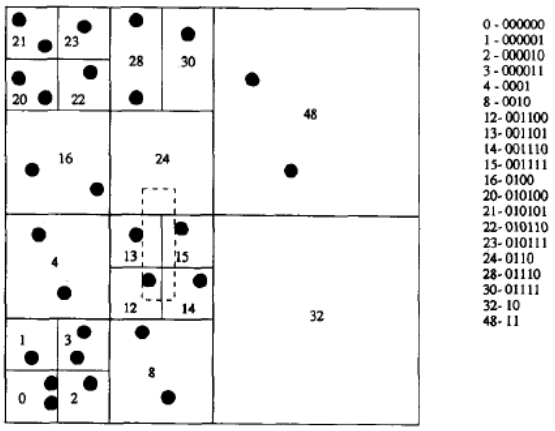


Figure 7.11: An example of partitioning for several data points.

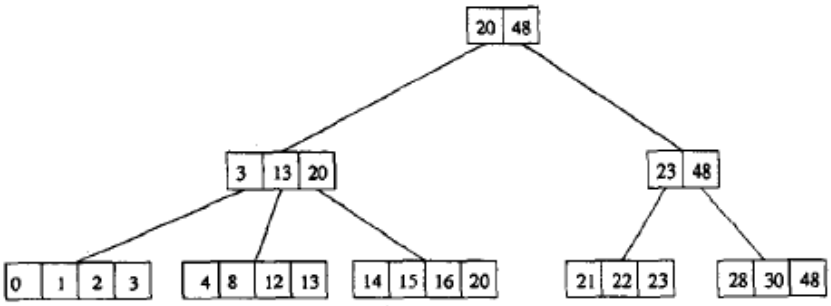


Figure 7.12: G-tree for the partitions of Fig.7.11

Range Query: The basic strategy consists of first identifying the smallest and largest partition numbers that could overlap with the query region. All partitions that lie within this range potentially contain points that overlap the query region. Next, the G-tree is searched and the partitions lying in this range are tested to determine whether they are fully contained in the query region, overlap the query region (but are not fully contained), or are outside the

query region. If a partition is fully contained, then all points in it satisfy the query. On the other hand, if it overlaps, then each point in it must be examined and checked individually. Finally, a partition that lies outside the query region does not have to be considered any further.

7.8 R TREE

In theoretical studies, we often develop structures that are dedicated to specific problems. In practice, often it is unrealistic to create many different indexes on the same dataset to support various types of queries, because doing so will incur prohibitive space consumption and update overhead. Therefore, it would be nice to have a single, all-around, structure, which occupies small space, can be updated efficiently, and most importantly, supports a large variety of queries.

Here we will discuss such an all-around structure called the R-tree. This structure is heuristic in nature, because it does not have any attractive theoretical guarantees on the search performance. Nevertheless, the practical efficiency of this structure has been widely established for many problems, especially if the dimensionality is low. Interestingly, for realistic datasets, there has been evidence that R-trees even outperform some theoretical worst-case efficient structures. The design of a theoretical structure aims at handling the most adverse datasets. Much of the design is not really needed for "good" datasets, and thus, may actually cause unnecessary overhead on such data. Because of its superb efficiency, the R-tree has become the de facto structure for multi-dimensional indexing in database systems nowadays. Our discussion is based on 2d point data, but the extensions to rectangle data and higher-dimensionalities are straightforward.

Let P be a set of points. An R-tree stores all these points in leaf nodes, each of which contains $\Theta(B)$ points, where B is the size of a disk page. Each non-leaf node u has $\Theta(B)$ children, except the root which must have 2 children at minimum unless it is the only node in the tree. For each child v, u stores a ***minimum bounding rectangle*** (MBR), which is the smallest rectangle that tightly encloses all the data points in the subtree of v. Note that there is no constraint on how points should be grouped into leaf nodes, and in general, how non-leaf nodes should be grouped into nodes of higher levels. Since each point is stored only once, the entire tree consumes linear space O(N/B), where N is the cardinality of P.

An R-Tree satisfies the following properties:

1. Every leaf node contains between m and M index records unless it is the root Thus, the root can have less entries than m
2. For each index record in a leaf node, I is the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple
3. Every non-leaf node has between m and M children unless it is the root
4. For each entry in a non-leaf node, i is the smallest rectangle that spatially contains the rectangles in the child node
5. The root node has at least two children unless it is a leaf
6. All leaves appear on the same level. That means the tree is balanced

Figure shows an example where P has 13 points p1, p2, ..., p13. Points p1, p2, p3, for example, are grouped into leaf node u1. This leaf is a child of of non-leaf node u6, which stores an MBR r1 for u1. Note that r1 tightly bounds all the points in u1.

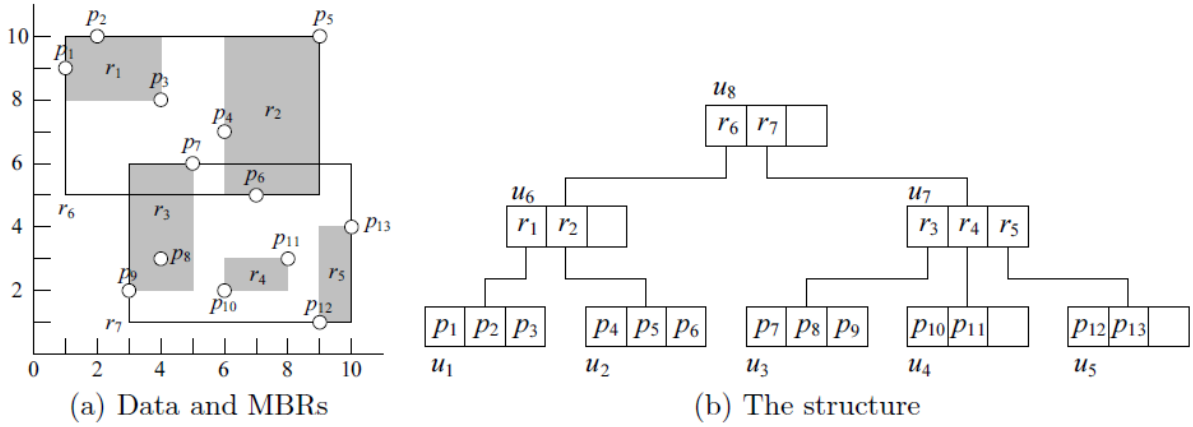


Figure 7.10: An R-tree

Insertions and deletions: Intuitively, in a good R-tree, nodes should have small MBRs. To see this intuitively, think about how to use the tree in Figure given above to answer a range query. Namely, given a query rectangle q, we want to find all the points in P that are covered by q. It is easy to see that we only have to visit those nodes whose MBRs intersect q. Therefore, reducing the extents of the MBRs benefits query efficiency as fewer MBRs are expected to intersect q.

Insertions: To insert a point p, we use a strategy similar to that of a B-tree. Specifically, we add p to a leaf node u by following a single root-to-leaf path. If u overflows, split it, which creates a new child of parent(u). In case parent(u) overflows, also split it, which propagates upwards in the same manner. Finally, if the root is split, then a new root is created.

While all these sound familiar, there are, however, two important differences. First, although in the B-tree the insertion path is unique (i.e., the leaf supposed to accommodate the new item is unambiguous), this is not true at all for the R-tree. In fact, the new point p can be inserted into any leaf, which always results in a legal structure. If, however, a bad leaf is chosen (to contain p), its MBR may need to be enlarged, thus harming the efficiency of the tree. Second, the split algorithm is not as trivial as in a B-tree because now we have multiple dimensions to tackle. Next, we will deal with the two issues separately. Note that the (heuristic) strategies to be introduced are not the only ones. In fact, this is why there are so many variants of R-trees – each of them has its own strategies.

Choosing a subtree to insert. We are essentially facing the following problem. Given a non-leaf node u with children v1, v2, ..., vf for some f = Θ(B), we need to pick the best child v^* such that the new point p is best inserted into the subtree of v^*. An approach that seems to work well in practice is a greedy one. Specifically, v^* can simply be the child v_i whose MBR requires the least increase of perimeter in order to cover p. For example, in Figure given below, both MBRs r1 and r2 must be expanded to enclose p, but r2 incurs smaller perimeter increase, and hence, is a better choice.

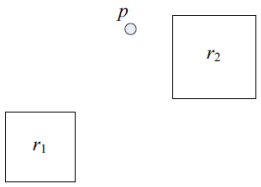


Figure: MBR r2 requires smaller perimeter increase to cover p

It is possible that p falls into the overlapping region of multiple MBRs. All those MBRs have tie because none of them needs any perimeter increase to cover p. In this case, the winner can be decided according to other factors such as picking the MBR having the smallest area.

Node split: The node split problem can be phrased as follows. Given a set S of B + 1 points, split it into disjoint subsets S1 and S2 with S1 [S2 = S such that

- $|S1| \geq \lambda B$, $|S2| \geq \lambda B$, where constant λ is the minimum utilization rate of a node, and
- the sum of the perimeters of MBR(S1) and MBR(S2) is small.

In the sequel, for simplicity we assume that |S| is an even number, and |S1| = |S2| = |S|/2, i.e., we always aim at an even split. The extensions to uneven splits are straightforward.

Ideally, we would like to find the optimal split that minimizes the perimeter sum of MBR(S1) and MBR(S2). Since an MBR is decided by 4 coordinates (i.e., a pair of opposite corners), it

is easy to find the optimal split in O(B4) time. This can be significantly improved to O(B2) time using a trick in [3]. Unfortunately, even a quadratic split time is usually excessively long in practice. Therefore, we turn our attention to heuristics that do not guarantee optimality, but usually produce fairly good splits. Next, we will describe a split algorithm that runs in O(B logB) time, or O(dB logB) time in general d-dimensional space.

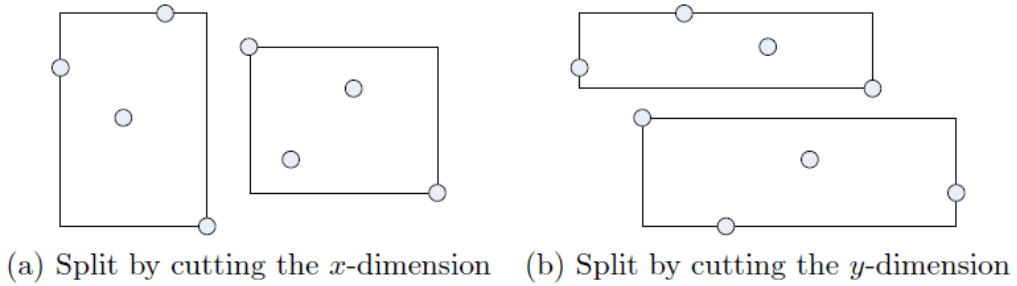


Figure 7.11: Splitting a node

The idea of our algorithm is to always split S using an axis-orthogonal cut. Consider, for example, a cut along the x-axis. For this purpose, we sort the points of S in ascending order of their x-coordinates. Then, we put the first B/2 points in the sorted order into S1, and the rest into S2. The split along the y-axis can be obtained in the same way. See Figure given above (where S has 8 points). The final split is the better one of the two splits.

The above applies to splitting a leaf node. The case of non-leaf node is a bit different because the items to be split are MBRs, as opposed to points. Nevertheless, similar heuristics can still be applied by, for example, sorting the MBRs by their centrioids along each dimension.

Deletions: Deleting a point from an R-tree is carried out in an interesting manner. In particular, node underflows are handled in a way that differs considerably from the conventional merging approach as in a B-tree.

Specifically, let p be the point to be deleted. First, we need to find the leaf node u where p is stored. This can be achieved with a special range query using p itself as the search region. Then, p is removed from u. The deletion finishes if u still has λB items, where λ denotes the minimum node utilization. Otherwise, u underflows, which is handled by first removing u from its parent, and then re-inserting all the remaining points in u (using exactly the insertion algorithm mentioned earlier). See Figure given below.

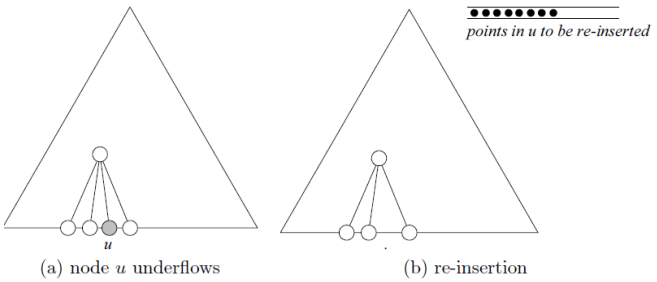


Figure 7.12: Handling a node underflow

Note that removing u from parent(u) may cause parent(u) to underflow too. In general, the underflow of a non-leaf node u' is also handled by re-insertions, with the only difference that the items re-inserted are MBRs, and each MBR is re-inserted to the same level of u'.

It is worth mentioning that while we can also design merging-based algorithms to handle node underflows, re-insertion actually gives better search performance [2]. This is because the structure of an R-tree is sensitive to the insertion order of the data points. Re-insertion gives the early-inserted points to be inserted in other (better) branches, thus improving the overall structure.

7.9 SUMMARY

In this unit, we have presented various data structures used in information retrieval systems. The most common data structure called inverted file structure is introduced in the beginning followed by N-gram data structures. The advanced data structures namely B-tree, B+-tree, KD tree, G Tree and R Tree useful for indexing are presented with a focus on query operations. The insertion and deletion algorithms are also given considering the generic structure.

7.10 KEYWORDS

Inverted files, N-gram data structure, B-tree, B+-tree, G Tree, R Tree, KD Tree, Indexing, Insertion, Deletion

7.11 QUESTIONS

1. Explain how inverted files are implemented using arrays with an example.
2. List the merits of inverted files
3. Discuss the structures used in inverted files.
4. What are n-gram data structures? Explain.

5. Define B-tree and height of B-tree
6. Discuss the procedure of searching an element in a B-tree.
7. Discuss the procedure of inserting an element into a B-tree.
8. Define B+-tree and height of B+-tree
9. Discuss the procedure of searching an element in a B+-tree.
10. Discuss the procedure of inserting an element into a B+-tree.
11. What are KD trees? Explain range searching in KD trees.
12. Describe the procedure of searching an element in a Kd tree and analyze its time complexity.
13. Define G tree and discuss the partitioning scheme of G tree.
14. What are R trees? Name the properties of R Trees.
15. With a suitable example, explain the process of inserting an element into a R tree.

7.12 REFERENCES FOR FURTHER READING/STUDIES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to Algorithms, Prentice Hall India, New Delhi.

[2] Franco P. Preparata and Micheal Ian Shamos, Computational Geometry - An Introduction, Springer-Verlag,USA.

[3] L. Arge, M. de Berg, H. J. Haverkort, and K. Yi. The priority R-tree: A practically efficient and worst-case optimal R-tree. In Proceedings of ACM Management of Data (SIGMOD), pages 347–358, 2004.

[3] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In Proceedings of ACM Management of Data (SIGMOD), pages 322–331, 1990.

[5] Y. J. Garcia, M. A. Lopez, and S. T. Leutenegger. On optimal node splitting for r-trees. In Proceedings of Very Large Data Bases (VLDB), pages 334–344, 1998.

[6] A. Guttman. R-trees: a dynamic index structure for spatial searching. In Proceedings of ACM Management of Data (SIGMOD), pages 47–57, 1984.

[7] I. Kamel and C. Faloutsos. Hilbert R-tree: An improved R-tree using fractals. In Proceedings of Very Large Data Bases (VLDB), pages 500–509, 1994.

[8] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A dynamic index for multi-dimensional objects. In Proceedings of Very Large Data Bases (VLDB), pages 507–518, 1987.

[9] Organization and Maintenance of Large Ordered Indexes by R. Bayer and E. McCreight In: Acta Informatica, Vol. 1, Fasc. 3, 1972, pp. 173-189.

[10] A Practical Introduction to Data Structures and Algorithm Analysis, 2nd edition, 2001, by Clifford A. Shaffer. Published by Pearson Prentice Hall,

UNIT 8: AUTOMATIC INDEXING

Structure

8.0 INTRODUCTION

Automatic indexing is the capability for the system to automatically determine the index terms to be assigned to an item. The simplest case is when all words in the document are used as possible index terms (total document indexing). More complex processing is required when the objective is to emulate a human indexer and determine a limited number of index terms for the major concepts in the item. The advantages of human indexing are the ability to determine concept abstraction and judge the value of a concept. The disadvantages of human indexing over automatic indexing are cost, processing time and consistency. Once the initial hardware cost is amortized, the costs of automatic indexing are absorbed as part of the normal

operations and maintenance costs of the computer system. There are no additional indexing costs versus the salaries and benefits regularly paid to human indexers. In this unit, main focus is on the process and algorithms to perform indexing. Each document is described by a set of features. Each class is described using the same kind of features. A document is associated to the class (es) where the features are most similar. This can be tested using rules or similarity measures. This unit deals with classes of Automatic Indexing, Statistical Indexing, Natural Language, Concept Indexing, and Hypertext Linkages.

8.1 CLASSES OF AUTOMATIC INDEXING

Automatic indexing is the process of analyzing an item to extract the information to be permanently kept in an index. This process is associated with the generation of the searchable data structures associated with an item.

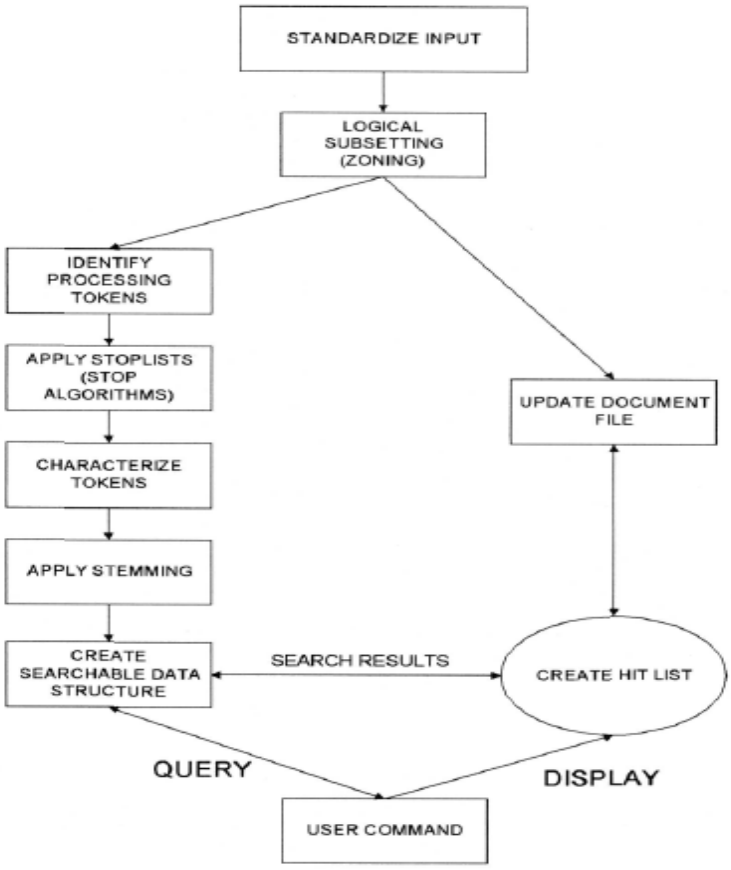


Figure 8.1 Data Flow in Information Processing System

The Data Flow diagram, in figure 8.1, shows where the indexing process is in the overall processing of an item. The figure also shows where the search process relates to the indexing

process. The left side of the figure including Identify Processing Tokens, Apply Stop Lists, Characterize tokens, Apply Stemming and Create Searchable Data Structure is all part of the indexing process. All systems go through an initial stage of zoning and identifying the processing tokens used to create the index. Some systems automatically divide the document up into fixed length passages or localities, which become the item unit that is indexed (Kretser-99.) Filters, such as stop lists and stemming algorithms, are frequently applied to reduce the number of tokens to be processed.

There is a major dependency between the search techniques to be implemented and the indexing process that stores the information required to execute the search. An index is the data structure created to support the search strategy. The different types of search or indexing strategies are listed below

1. Statistical strategies
2. Natural language strategies
3. Concept indexing
4. Hypertext linkages.

Statistical strategies cover the broadest range of indexing techniques and are the most prevalent in commercial systems. The basis for a statistical approach is use of frequency of occurrence of events. Natural Language approaches perform the similar processing token identification as in statistical techniques, but then additionally perform varying levels of natural language parsing of the item. This parsing disambiguates the context of the processing tokens and generalizes to more abstract concepts within an item (e.g., present, past, future actions). Concept indexing uses the words within an item to correlate to concepts discussed in the item. This is a generalization of the specific words to values used to index the item. A special class of indexing can be defined by creation of hypertext linkages. These linkages provide virtual threads of concepts between items versus directly defining the concept within an item. All these methods are discussed in detail in the following sections.

8.2 STATISTICAL INDEXING

Statistical strategies cover the broadest range of indexing techniques and are the most prevalent in commercial systems. The basis for a statistical approach is use of frequency of occurrence of events. The events usually are related to occurrences of processing tokens (words/phrases) within documents and within the database. The words/phrases are the domain of searchable values. The statistics that are applied to the event data are probabilistic,

Bayesian, vector space, neural network. The static approach stores a single statistic, such as how often each word occurs in an item that is used in generating relevance scores after a standard Boolean search. The probabilistic indexing stores the information that are used in calculating a probability that a particular item satisfies (i.e., is relevant to) a particular query. A probability of 50 per cent would mean that if enough items are reviewed, on the average one half of the reviewed items are relevant. Bayesian and vector approaches store information used in generating a relative confidence level of an item's relevance to a query. Neural networks are dynamic learning structures that are discussed under concept indexing where they are used to determine concept classes.

8.2.1 Probabilistic Weighting

The probabilistic approach is based upon direct application of the theory of probability to information retrieval systems. This has the advantage of being able to use the developed formal theory of probability to direct the algorithmic development. It also leads to an invariant result that facilitates integration of results from different databases. The use of probability theory is a natural choice because it is the basis of evidential reasoning (i.e., drawing conclusions from evidence).

HYPOTHESIS: If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of usefulness to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data is available for this purpose, then the overall effectiveness of the system to its users is the best obtainable on the basis of that data.

PLAUSIBLE COROLLARY: The most promising source of techniques for estimating the probabilities of usefulness for output ranking in IR is standard probability theory and statistics.

Probabilities are usually based upon a binary condition; an item is relevant or not. But in information systems the relevance of an item is a continuous function from non-relevant to absolutely useful. To address this characteristic a more complex theory of expected utility (Cooper-78) is needed. The source of the problems that arise in application of probability theory come from a lack of accurate data and simplifying assumptions that are applied to the mathematical model. If nothing else, these simplifying assumptions cause the results of probabilistic approaches in ranking items to be less accurate than other approaches. The advantage of the probabilistic approach is that it can accurately identify its weak assumptions

and work to strengthen them. In many other approaches, the underlying weaknesses in assumptions are less obvious and harder to identify and correct.

There are many different areas in which the probabilistic approach may be applied. The method of logistic regression is described as an example of how a probabilistic approach is applied to information retrieval (Gey-94). The approach starts by defining a "Model 0" system which exists before specific probabilistic models are applied. In a retrieval system there exist query terms **qi** and document terms **di** which have a set of attributes **(V1, V2, ...Vn)** from the query (e.g., counts of term frequency in the query), from the document (e.g., counts of term frequency in the document) and from the database (e.g., total number of documents in the database divided by the number of documents indexed by the term).

The logistic reference model uses a random sample of query-document term triples for which binary relevance judgments have been made from a training sample. Log O is the logarithm of the odds (logodds) of relevance for term **tk** which is present in document **Dj** and query **qi.**

$$\log(O(R \mid Q_i, D_j, t_k)) = c_0 + c_1 v_1 + \ldots + c_n v_n$$

The logarithm that the ith Query is relevant to the jth Document is the sum of the logodds for all terms:

$$\log(O(R \mid Q_i, D_j)) = \sum_{k=1}^{q} [\log(O(R \mid Q_i, D_j, t_k)) - \log(O(R))]$$

where O(R) is the odds that a document chosen at random from the database is relevant to query **Qi**. The coefficients ***ci*** are derived using logistic regression which fits an equation to predict a dichotomous independent variable as a function of independent variables that show statistical variation (Hosmer-89). The inverse logistic transformation is applied to obtain the probability of relevance of a document to a query:

$$P(R \mid Q_i, D_j) = 1 \setminus (1 + e^{-\log(O(R \mid Q_i, D_j))})$$

The coefficients of the equation for logodds is derived for a particular database using a random sample of query-document-term-relevance quadruples and used to predict odds of relevance for other query-document pairs.

Gey applied this methodology to the Cranfield Collection (Gey-94). The collection has 1400 items and 225 queries with known results. Additional attributes of relative frequency in the

query (QRF), relative frequency in the document (DRF) and relative frequency of the term in all the documents (RFAD) were included, producing the following logodds formula

$$Z_j = \log(O(R \mid t_j)) = c_0 + c_1\log(QAF) + c_2\log(QRF) + c_3\log(DAF) + c_4\log(DRF) + c_5\log(IDF) + c_6\log(RFAD)$$

where QAF, DAF, and IDF were previously defined.

QRF = QAF\ (total number of terms in the query),

DRF = DAF\(total number of words in the document) and

RFAD = (total number of term occurrences in the database)\ (total number of all words in the database).

Logs are used to reduce the impact of frequency information; then smooth out skewed distributions. A higher maximum likelihood is attained for logged attributes. The coefficients and log (O(R)) were calculated creating the final formula for ranking for query vector $\vec{Q}$ which contains q terms:

$$\log(O(R \mid \vec{Q})) = -5.138 + \sum_{k=1}^{q} (Z_j + 5.138)$$

The logistic inference method was applied to the test database along with the Cornell SMART vector system which uses traditional term frequency, inverse document frequency and cosine relevance weighting formulas. The logistic inference method outperformed the vector method. Thus the index that supports the calculations for the logistic reference model contains the O(R) constant value (e.g., -5.138) along with the coefficients through additionally, it needs to maintain the data to support DAF, DRF, IDF and RFAD. The values for QAF and QRF are derived from the query.

Attempts have been made to combine the results of different probabilistic techniques to get a more accurate value. The objective is to have the strong points of different techniques compensate for weaknesses. To date this combination of probabilities using averages of Log-Odds has not produced better results and in many cases produced worse results.

8.2.2. Vector Weighting

One of the earliest systems that investigated statistical approaches to information retrieval was the SMART system at Cornell University (Buckley-95, Salton-83). The system is based

upon a vector model. The semantics of every item are represented as a vector. A vector is a one-dimensional set of values, where the order/position of each value in the set is fixed and represents a particular domain. In information retrieval, each position in the vector typically represents a processing token. There are two approaches to the domain of values in the vector: binary and weighted. Under the binary approach, the domain contains the value of one or zero, with one representing the existence of the processing token in the item. In the weighted approach, the domain is typically the set of all real positive numbers. The value for each processing token represents the relative importance of that processing token in representing the semantics of the item. Figure 5.2 shows how an item that discusses petroleum refineries in Mexico would be represented. In the example, the major topics discussed are indicated by the index terms for each column (i.e., Petroleum, Mexico, Oil, Taxes, Refineries and Shipping).

Binary vectors require a decision process to determine if the degree that a particular processing token represents the semantics of an item is sufficient to include it in the vector. In the example for Figure 8.2, a five-page item may have had only one sentence like "Standard taxation of the shipment of the oil to refineries is enforced." For the binary vector, the concepts of "Tax" and "Shipment" are below the threshold of importance (e.g., assume threshold is 1.0) and they not are included in the vector.

	Petroleum	Mexico	Oil	Taxes	Refineries	Shipping
Binary	(1 ,	1 ,	1 ,	0 ,	1 ,	0)
Weighted	(2.8 ,	1.6 ,	3.5 ,	.3 ,	3.1 ,	.1)

Figure 8.2 Binary and Vector Representation of an Item

A weighted vector acts the same as a binary vector but it provides a range of values that accommodates a variance in the value of the relative importance of a processing token in representing the semantics of the item. The use of weights also provides a basis for determining the rank of an item. The vector approach allows for a mathematical and a physical representation using a vector space model. Each processing token can be considered another dimension in an item representation space. Figure 5.3 shows a three-dimensional vector representation assuming there were only three processing tokens, Petroleum Mexico and Oil.

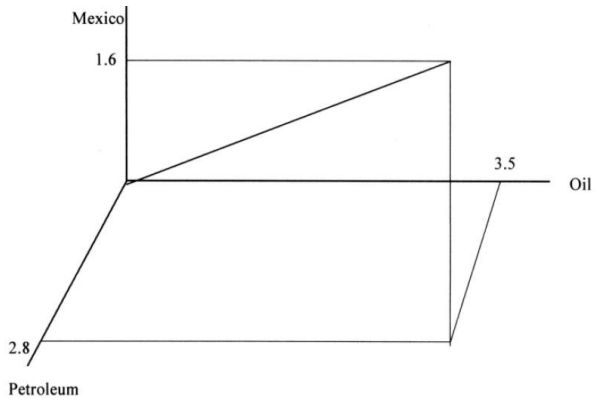


Figure 8.3 Vector Representation

The original document vector has been extended by additional information such as citations/references to add more information for search and clustering purposes. There have not been significant improvements in retrieval using these techniques. Introduction of text generated from multimedia sources introduces a new rationale behind extending the vocabulary associated with an item. In the case where the text is not generated directly by an author but is the result of audio transcription, the text will contain a significant number of word errors. These will be valid words but the wrong word. One mechanism to reduce the impact of the missing words is to use the existing database to expand the document. This is accomplished by using the transcribed document as a query against the existing database, selecting a small number of the highest ranked results, determining the most important (highest frequency) words across those items and adding those words to the original document. The new document will then be normalized and reweighted based upon the added words (Singhal-99). The following subsections present the major algorithms that can be used in calculating the weights, used to represent a processing token starting with the most simple term frequency algorithm.

8.2.2.1 Simple Term Frequency Algorithm

In both the unweighted and weighted approaches, an automatic indexing process implements an algorithm to determine the weight to be assigned to a processing token for a particular item. In a statistical system, the data that are potentially available for calculating a weight are the frequency of occurrence of the processing token in an existing item (i.e., term frequency - TF), the frequency of occurrence of the processing token in the existing database (i.e., total frequency - TOTF) and the number of unique items in the database that contain the processing token (i.e., item frequency - IF, frequently labeled in other publications as document frequency - DF). The premises by Luhn and later Brookstein states that the

resolving power of content-bearing words is directly proportional to the frequency of occurrence of the word in the item is used as the basis for most automatic weighting techniques. Weighting techniques usually are based upon positive weight values.

The simplest approach is to have the weight equal to the term frequency. This approach emphasizes the use of a particular processing token within an item. Thus if the word "computer" occurs 15 times within an item it has a weight of 15. The simplicity of this technique encounters problems of normalization between items and use of the processing token within the database. The longer an item is, the more often a processing token may occur within the item. Use of the absolute value biases weights toward longer items, where a term is more likely to occur with a higher frequency. Thus, one normalization typically used in weighting algorithms compensates for the number of words in an item. An example of this normalization in calculating term-frequency is the algorithm used in the SMART System at Cornell (Buckley-96). The term frequency weighting formula used in TREC 4 was:

$$\frac{(1 + \log(TF))/1 + \log(\text{average } (TF))}{(1 - \text{slope}) * \text{pivot} + \text{slope} * \text{number of unique terms}}$$

where slope was set at .2 and the pivot was set to the average number of unique terms occurring in the collection (Singhal-95). In addition to compensating for document length, they also want the formula to be insensitive to anomalies introduced by stemming or misspellings.

Although initially conceived of as too simple, recent experiments by the SMART system using the large databases in TREC demonstrated that use of the simpler algorithm with proper normalization factors is far more efficient in processing queries and return hits similar to more complex algorithms.

There are many approaches to account for different document lengths when determining the value of Term Frequency to use (e.g., an items that is only 50 words may have a much smaller term frequency then and item that is 1000 words on the same topic). In the first technique, the term frequency for each word is divided by the maximum frequency of the word in any item. This normalizes the term frequency values to a value between zero and one. This technique is called maximum term frequency. The problem with this technique is that the maximum term frequency can be so large that it decreases the value of term frequency in short items to too small a value and loses significance.

Another option is to use logaritmetic term frequency. In this technique the log of the term frequency plus a constant is used to replace the term frequency. The log function will perform the normalization when the term frequencies vary significantly due to size of documents. Along this line the COSINE function used as a similarity measure can be used to normalize values in a document. This is accomplished by treating the index of a document as a vector and divides the weights of all terms by the length of the vector. This will normalize to a vector of maximum length one. This uses all of the data in a particular item to perform the normalization and will not be distorted by any particular term. The problem occurs when there are multiple topics within an item. The COSINE technique will normalize all values based upon the total length of the vector that represents all of topics. If a particular topic is important but briefly discussed, its normalized value could significantly reduce its overall importance in comparison to another document that only discusses the topic.

Another approach recognizes that the normalization process may be over penalizing long documents (Singhal-95). Singhal did experiments that showed longer documents in general are more likely to be relevant to topics then short documents. Yet normalization was making all documents appear to be the same length. To compensate, a correction factor was defined that is based upon document length that maps the Cosine function into an adjusted normalization function. The function determines the document length crossover point for longer documents where the probability of relevance equals the probability of retrieval, (given a query set). This value called the "pivot point" is used to apply an adjustment to the normalization process. The theory is based upon straight lines so it is a matter of determining slope of the lines.

$$\text{New normalization} = (\text{slope})*(\text{old normalization}) + K$$

K is generated by the rotation of the pivot point to generate the new line and the old normalization = the new normalization at that point. The slope for all higher values will be different. Substituting pivot for both old and new value in the above formula we can solve for K at that point. Then using the resulting formula for K and substituting in the above formula produces the following formula:

$$\text{Pivoted function} = \text{slope})*(\text{old normalization}) + (1.0 - \text{slope})*(\text{pivot})$$

Slope and pivot are constants for any document/query set. Another problem is that the Cosine function favours short documents over long documents and also favours documents with a

large number of terms. This favouring is increased by using the pivot technique. If log(TF) is used instead of the normal frequency then TF is not a significant factor. In documents with large number of terms the Cosine factor is approximated by the square root of the number of terms. This suggests that using the ratio of the logs of term frequencies would work best for longer items in the calculations:

$$(1 + \log(TF))/(1 + \log(\text{average}(TF))$$

This leads to the final algorithm that weights each term by the above formula divided by the pivoted normalization. Singhal demonstrated the above formula works better against TREC data then TF/MAX(TF) or vector length normalization. The effect of a document with a high term frequency is reduced by the normalization function by dividing the TF by the average TF and by use of the log function. The use of pivot normalization

$$((1 + \log(TF))/(1 + \log(\text{average}(TF))/(\text{slope})(\text{No. unique terms}) + (1\text{-slope})*(\text{pivot})$$

Singhal demonstrated the above formula works better against TREC data then TF/MAX(TF) or vector length normalization. The effect of a document with a high term frequency is reduced by the normalization function by dividing the TF by the average TF and by use of the log function. The use of pivot normalization adjusts for the bias towards shorter documents increasing the weights of longer documents.

8.2.2.2 Inverse Document Frequency

The basic algorithm is improved by taking into consideration the frequency of occurrence of the processing token in the database. One of the objectives of indexing an item is to discriminate the semantics of that item from other items in the database. If the token "computer" occurs in every item in the database, its value representing the semantics of an item may be less useful compared to a processing token that occurs in only a subset of the items in the database. The term "computer" represents a concept used in an item, but it does not help a user find the specific information being sought since it returns the complete database. This leads to the general statement enhancing weighting algorithms that the weight assigned to an item should be inversely proportional to the frequency of occurrence of an item in the database. This algorithm is called inverse document frequency (IDF). The un-normalized weighting formula is:

$$WEIGHT_{ij} = TF_{ij} * [\text{Log}_2(n) - \text{Log}_2(IF_j) + 1]$$

where WEIGHTij is the vector weight that is assigned to term "j" in item "i," **TFij(**term frequency) is the frequency of term "j" in item "i" , "n" is the number of items in the database and **IFj(**item frequency or document frequency) is the number of items in the database that have term "j" in them. A negative log is the same as dividing by the log value, thus the basis for the name of the algorithm.

Figure 8.4 demonstrates the impact of using this weighting algorithm. The term "refinery" has the highest frequency in the new item (10 occurrences). But it has a normalized weight of 20 which is less than the normalized weight of "Mexico." This change in relative importance between "Mexico" and "refinery" from the unnormalized to normalized vectors is due to an adjustment caused by "refinery" already existing in 50 per cent of the database versus "Mexico" which is found in 6.25 per cent of the items.

The major factor of the formula for a particular term is $(\mathrm{Log_2(n)} - \mathrm{Log_2(IF_j)})$. The value for IF can vary from "1" to "n." At "n," the term is found in every item in the database and the factor becomes $(\mathrm{Log_2(n)} - \mathrm{Log_2(n)}) = 1$. As the number of items a term is found in decreases, the value of the denominator decreases eventually approaching the value $\mathrm{Log_2(1)}$ which is close to 1. The weight assigned to the term in the item varies from $\mathrm{Tf_{i,j}} * (1 + 1)$ to $\mathrm{Tf_{i,j}} * (\sim \mathrm{Log_2(n)})$. The effect of this factor can be too great as the number of items that a term is found in becomes small. To compensate for this, the INQUERY system at the University of Massachusetts normalizes this factor by taking an additional log value.

Assume that the term "oil" is found in 128 items, "Mexico" is found in 16 items and "refinery" is found in 1024 items. If a new item arrives with all three terms in it, "oil" found 4 times, "Mexico" found 8 times, and "refinery found 10 times and there are 2048 items in the total database, Figure 4.4 shows the weight calculations using inverse document frequency.

Using a simple unnormalized term frequency, the item vector is (4, 8, 10) Using inverse document frequency the following calculations apply with the resultant inverse document frequency item vector = (20, 64, 20).

$$Weight_{oil} = 4 * (Log_2(2048) - Log_2(128) + 1) = 4 * (11 - 7 + 1) = 20$$

$$Weight_{Mexico} = 8 * (Log_2(2048) - Log_2(16) + 1) = 8 * (11 - 4 + 1) = 64$$

$$Weight_{refinery} = 10 * (Log_2(2048) - Log_2(1024) + 1) = 10 * (11 - 10 + 1) = 20$$

Figure 8.4 Example of Inverse Document Frequency

The value of "n" and **IFi** vary as items are added and deleted from the database. To implement this algorithm in a dynamically changing system, the physical index only stores the frequency of occurrence of the terms in an item (usually with their word location) and the IDF factor is calculated dynamically at retrieval time. The required information can easily be determined from an inversion list for a search term that is retrieved and a global variable on the number of items in the database.

8.2.2.3 Signal Weighting

Inverse document frequency adjusts the weight of a processing token for an item based upon the number of items that contain the term in the existing database. What it does not account for is the term frequency distribution of the processing token in the items that contain the term. The distribution of the frequency of processing tokens within an item can affect the ability to rank items.

For example, assume the terms "SAW" and "DRILL" are found in 5 items with the following frequencies defined in Figure 4.5. Both terms are found a total of 50 times in the five items. The term "SAW" does not give any insight into which item is more likely to be relevant to a search of "SAW". If precision is a goal (maximizing relevant items shown first), then the weighting algorithm could take into consideration the non-uniform distribution of term "DRILL" in the items that the term is found, applying even higher weights to it than "SAW." The theoretical basis for the algorithm to emphasize precision is Shannon's work on Information Theory (Shannon-51).

In Information Theory, the information content value of an object is inversely proportional to the probability of occurrence of the item. An instance of an event that occurs all the time has less information value than an instance of a seldom occurring event.

Item Distribution	SAW	DRILL
A	10	2
B	10	2
C	10	18
D	10	10
E	10	18

Figure 8.5 Item Distributions for SAW and DRILL

This is typically represented as INFORMATION = -Log2 (p), where p is the probability of occurrence of event "p." The information value for an event that occurs .5 per cent of the time is:

$$\begin{aligned} \text{INFORMATION} &= -\text{Log}_2(.0005) \\ &= -(-10) \\ &= 10 \end{aligned}$$

The information value for an event that occurs 50 per cent of the time is:

$$\begin{aligned} \text{INFORMATION} &= -\text{Log}_2(.50) \\ &= -(-1) \\ &= 1 \end{aligned}$$

If there are many independent occurring events then the calculation for the average information value across the events is:

$$\text{AVE\_INFO} = -\sum_{k=1}^{n} p_k \text{Log}_2(p_k)$$

The value of AVE_INFO takes its maximum value when the values for every **pk** are the same. Its value decreases proportionally to increases in variances in the values of **pk.** The value of **pk can** be defined as the ratio of the frequency of occurrence of the term in an item to the total number of occurrences of the item in the data base. Using the AVE_INFO formula, the terms that have the most uniform distribution in the items that contain the term have the maximum value. To use this information in calculating a weight, the formula needs the inverse of AVE_INFO, where the minimum value is associated with uniform distributions and the maximum value is for terms that have large variances in distribution in the items containing the term. The following formula for calculating the weighting factor called Signal (Dennis-67) can be used:

$$Signal_k = Log_2 (TOTF) - AVE\_INFO$$

Producing a final formula of:

$$Weight_{ik} = TF_{ik} * Signal_k$$

$$Weight_{ik} = TF_{ik} * [Log_2(TOTF_k) - \sum_{i=1}^{n} TF_{ik}/TOTF_k \, Log_2 (TF_{ik}/TOTF_k)]$$

An example of use of the weighting factor formula is given for the values in Figure 4.5:

$$Signal_{SAW} = LOG_2 (50) - [5 * \{10/50LOG_2(10/50)\}]$$

$$Signal_{DRILL} = LOG_2 (50) - [2/50LOG_2(2/50) + 2/50LOG_2(2/50) + 18/50LOG_2(18/50) + 10/50LOG_2(10/50) + 18/50LOG_2(18/50)$$

The weighting factor for term "DRILL" that does not have a uniform distribution is larger than that for term "SAW" and gives it a higher weight. This technique could be used by itself or in combination with inverse document frequency or other algorithms. The overhead of the additional data needed in an index and the calculations required to get the values have not been demonstrated to produce better results than other techniques and are not used in any systems at this time. It is a good example of use of Information Theory in developing information retrieval algorithms. Effectiveness of use of this formula can be found in results from Harman and also from Lockbaum and Streeter (Harman-86, Lochbaum-89).

8.2.2.4 Discrimination Value

Another approach to creating a weighting algorithm is to base it upon the discrimination value of a term. To achieve the objective of finding relevant items, it is important that the index discriminates among items. The more all items appear the same, the harder it is to identify those that are needed. Salton and Yang (Salton-73) proposed a weighting algorithm that takes into consideration the ability for a search term to discriminate among items. They proposed use of a discrimination value for each term "i":

$$DISCRIM_i = AVESIM_i - AVESIM$$

where AVESIM is the average similarity between every item in the database and AVESIMi is the same calculation except that term "i" is removed from all items. There are three possibilities with the DISCRIMi value being positive, close to zero or negative. A positive value indicates that removal of term "i" has increased the similarity between items. In this case, leaving the term in the database assists in discriminating between items and is of value.

A value close to zero implies that the term's removal or inclusion does not change the similarity between items. If the value of DISCRIMi is negative, the term's effect on the database is to make the items appear more similar since their average similarity decreased with its removal. Once the value of DISCRMi is normalized as a positive number, it can be used in the standard weighting formula as:

$$\mathbf{Weight_{ik} = TF_{ik} * DISCRIM_k}$$

8.2.2.5 Problems with Weighting Schemes

Often weighting schemes use information that is based upon processing token distributions across the database. The two weighting schemes, inverse document frequency and signal, use total frequency and item frequency factors which makes them dependent upon distributions of processing tokens within the database. Information databases tend to be dynamic with new items always being added and to a lesser degree old items being changed or deleted. Thus these factors are changing dynamically. There are a number of approaches to compensate for the constant changing values.

a. Ignore the variances and calculate weights based upon current values, with the factors changing over time. Periodically rebuild the complete search database.
b. Use a fixed value while monitoring changes in the factors. When the changes reach a certain threshold, start using the new value and update all existing vectors with the new value.
c. Store the invariant variables (e.g., term frequency within an item) and at search time calculate the latest weights for processing tokens in items needed for search terms.

In the first approach the assumption minimizes the system overhead of maintaining currency on changing values, with the effect that term weights for the same term vary from item to item as the aggregate variables used in calculating the weights based upon changes in the database vary over time. Periodically the database and all term weights are recalculated based upon the most recent updates to the database. For large databases in the millions of items, the overhead of rebuilding the database can be significant. In the second approach, there is recognition that for the most frequently occurring items, the aggregate values are large. As such, minor changes in the values have negligible effect on the final weight calculation. Thus, on a term basis, updates to the aggregate values are only made when sufficient changes not using the current value will have an effect on the final weights and the search/ranking

process. This process also distributes the update process over time by only updating a subset of terms at any instance in time. The third approach is the most accurate. The weighted values in the database, only matter when they are being used to determine items to return from a query or the rank order to return the items. This has more overhead in that database vector term weights must be calculated dynamically for every query term. If the system is using an inverted file search structure, this overhead is very minor.

An interesting side effect of maintaining currency in the database for term weights is that the same query over time returns a different ordering of items. A new word in the database undergoes significant changes in its weight structure from initial introduction until its frequency in the database reaches a level where small changes do not have significant impact on changes in weight values.

Another issue is the desire to partition an information database based upon time. The value of many sources of information vary exponentially based upon the age of an item (older items have less value). This leads to physically partitioning the database by time (e.g., starting a new database each year), allowing the user to specify the time period to search. There are issues then of how to address the aggregate variables that are different for the same processing token in each database and how to merge the results from the different databases into a single Hit file.

The best environment would allow a user to run a query against multiple different time periods and different databases that potentially use different weighting algorithms, and have the system integrate the results into a single ranked Hit file.

8.2.2.6 Problems with the Vector Model

In addition to the general problem of dynamically changing databases and the effect on weighting factors, there are problems with the vector model on assignment of a weight for a particular processing token to an item. Each processing token can be viewed as a new semantic topic. A major problem comes in the vector model when there are multiple topics being discussed in a particular item. For example, assume that an item has an in-depth discussion of "oil" in "Mexico" and also "coal" in "Pennsylvania." The vector model does not have a mechanism to associate each energy source with its particular geographic area. There is no way to associate correlation factors between terms (i.e., precoordination) since each dimension in a vector is independent of the other dimensions. Thus the item results in a high value in a search for "coal in Mexico."

Another major limitation of a vector space is in associating positional information with a processing term. The concept of proximity searching (e.g., term "a" within 10 words of term "b") requires the logical structure to contain storage of positional information of a processing term. The concept of a vector space allows only one scalar value to be associated with each processing term for each item. Restricting searches to subsets of an item has been shown to provide increased precision. In effect this capability overcomes the multitopical item problem by looking at subsets of an item and thus increasing the probability that the subset is discussing a particular semantic topic.

8.2.3 Bayesian Model

Bayesian approach can be used to overcome the restrictions inherent in a vector model. It provides a conceptually simple yet complete model for information systems. In its most general definition, the Bayesian approach is based upon conditional probabilities (e.g., Probability of Event 1 given Event 2 occurred). This general concept can be applied to the search function as well as to creating the index to the database. The objective of information systems is to return relevant items. Thus the general case, using the Bayesian formula, is (PREL/DOCi , Queryj) which is interpreted as the probability of relevance (REL) to a search statement given a particular document and query. In addition to search, Bayesian formulas can be used in determining the weights associated with a particular processing token in an item. The objective of creating the index to an item is to represent the semantic information in the item. A Bayesian network can be used to determine the final set of processing tokens (called topics) and their weights. Figure 4.6 shows a simple view of the process where ti represents the relevance of topic "i" in a particular item and pj represents a statistic associated with the event of processing token "j" being present in the item.

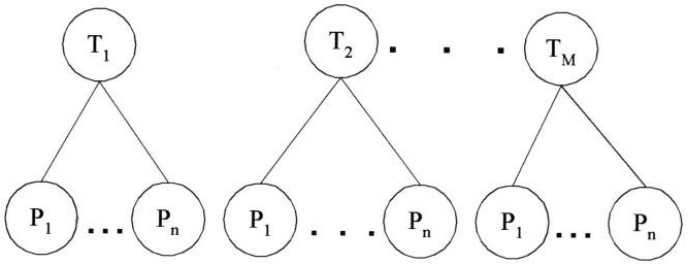


Figure 8.6 Bayesian Term Weighting

The "m" topics would be stored as the final index to the item. The statistics associated with the processing token are typically frequency of occurrence. But they can also incorporate

proximity factors that are useful in items that discuss multiple topics. There is one major assumption made in this model:

> **Assumption of Binary Independence:** the topics and the processing token statistics are independent of each other. The existence of one topic is not related to the existence of the other topics. The existence of one processing token is not related to the existence of other processing tokens.

In most cases this assumption is not true. Some topics are related to other topics and some processing tokens related to other processing tokens. For example, the topics of "Politics" and "Economics" are in some instances related to each other (e.g., an item discussing Congress debating laws associated with balance of trade) and in many other instances totally unrelated. The same type of example would apply to processing tokens. There are two approaches to handling this problem. The first is to assume that there are dependencies, but that the errors introduced by assuming the mutual independence do not noticeably effect the determination of relevance of neither an item nor its relative rank associated with other retrieved items. This is the most common approach used in system implementations. A second approach can extend the network to additional layers to handle interdependencies. Thus an additional layer of Independent Topics (ITs) can be placed above the Topic layer and a layer of Independent Processing Tokens (IPs) can be placed above the processing token layer. Figure 4.7 shows the extended Bayesian network. Extending the network creates new processing tokens for those cases where there are dependencies between processing tokens. The new set of Independent Processing Tokens can then be used to define the attributes associated with the set of topics selected to represent the semantics of an item. To compensate for dependencies between topics the final layer of Independent Topics is created. The degree to which each layer is created depends upon the error that could be introduced by allowing for dependencies between Topics or Processing Tokens. Although this approach is the most mathematically correct, it suffers from losing a level of precision by reducing the number of concepts available to define the semantics of an item.

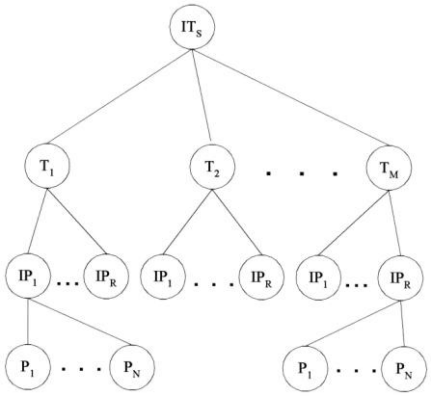


Figure 8.7 Extended Bayesian Network

8.3 NATURAL LANGUAGE

Natural Language approaches perform the similar processing token identification as in statistical techniques, but then additionally perform varying levels of natural language parsing of the item. This parsing disambiguates the context of the processing tokens and generalizes to more abstract concepts within an item (e.g., present, past, future actions). The goal of natural language processing is to use the semantic information in addition to the statistical information to enhance the indexing of the item. This improves the precision of searches, reducing the number of false hits a user reviews. The semantic information is extracted as a result of processing the language rather than treating each word as an independent entity. The simplest output of this process results in generation of phrases that become indexes to an item. More complex analysis generates thematic representation of events rather than phrases.

Statistical approaches use proximity as the basis behind determining the strength of word relationships in generating phrases. For example, with a proximity constraint of adjacency, the phrases “venetian blind” and “blind Venetian” may appear related and map to the same phrase. But syntactically and semantically those phrases are very different concepts. Word phrases generated by natural language processing algorithms enhance indexing specification and provide another level of disambiguation. Natural language processing can also combine the concepts into higher level concepts sometimes referred to as thematic representations.

8.3.1 Index Phrase Generation

The goal of indexing is to represent the semantic concepts of an item in the information system to support finding relevant information. Single words have conceptual context, but frequently they are too general to help the user find the desired information. Term phrases allow additional specification and focusing of the concept to provide better precision and

reduce the user's overhead of retrieving non-relevant items. Having the modifier "grass" or "magnetic" associated with the term "field" clearly disambiguates between very different concepts. One of the earliest statistical approaches to determining term phrases proposed by Salton was use of a COHESION factor between terms (Salton-83):

$$\text{COHESION}_{k,h} = \text{SIZE-FACTOR} * (\text{PAIR-FREQ}_{k,h} / \text{TOTF}_k * \text{TOTF}_H)$$

where SIZE-FACTOR is a normalization factor based upon the size of the vocabulary and $\text{PAIR-FREQ}_{k,h}$ is the total frequency of co-occurrence of the pair Termk , Termh in the item collection. Co-occurrence may be defined in terms of adjacency, word proximity, sentence proximity, etc. This initial algorithm has been modified in the SMART system to be based on the following guidelines (BUCKLEY-95):

1. Any pair of adjacent non-stop words is a potential phrase
2. Any pair must exist in 25 or more items
3. Phrase weighting uses a modified version of the SMART system single term algorithm
4. Normalization is achieved by dividing by the length of the single-term subvector.

Natural language processing can reduce errors in determining phrases by determining inter-item dependencies and using that information to create the term phrases used in the indexing process. Statistical approaches tend to focus on two term phrases. A major advantage of natural language approaches is their ability to produce multiple-term phrases to denote a single concept. If a phrase such as "industrious intelligent students" was used often, a statistical approach would create phrases such as "industrious intelligent" and "intelligent student." A natural language approach would create phrases such as "industrious student," "intelligent student" and "industrious intelligent student."

The first step in a natural language determination of phrases is a lexical analysis of the input. In its simplest form this is a part of speech tagger that, for example, identifies noun phrases by recognizing adjectives and nouns. Precise part of speech taggers exist that are accurate to the 99 per cent range. Additionally, proper noun identification tools exist that allow for accurate identification of names, locations and organizations since these values should be indexed as phrases and not undergo stemming. Greater gains come from identifying syntactic and semantic level dependencies creating a hierarchy of semantic concepts. For example, "nuclear reactor fusion" could produce term phrases of "nuclear reactor" and "nuclear fusion." In the ideal case all variations of a phrase would be reduced to a single canonical

form that represents the semantics for a phrase. Thus, where possible the phrase detection process should output a normalized form. For example, “blind Venetian” and “Venetian who is blind” should map to the same phrase. This not only increases the precision of searches, but also increases the frequency of occurrence of the common phrase. This, in turn, improves the likelihood that the frequency of occurrence of the common phrase is above the threshold required to index the phrase. Once the phrase is indexed, it is available for search, thus participating in an item’s selection for a search and the rank associated with an item in the Hit file. One solution to finding a common form is to transform the phrases into a operator-argument form or a header-modifier form. There is always a category of semantic phrases that comes from inferring concepts from an item that is non-determinable. This comes from the natural ambiguity inherent in languages.

A good example of application of natural language to phrase creation is in the natural language information retrieval system at New York University developed in collaboration with GE Corporate Research and Development (Carballo-95). The text of the item is processed by a fast syntactical process and extracted phrases are added to the index in addition to the single word terms. Statistical analysis is used to determine similarity links between phrases and identification of sub-phrases. Once the phrases are statistically noted as similar, a filtering process categorizes the link onto a semantic relationship (generality, specialization, antonymy, complementation, synonymy, etc.).

The Tagged Text Parser (TTP), based upon the Linguistic String Grammar (Sager-81), produces a regularized parse tree representation of each sentence reflecting the predicate-argument structure (Strzalkowski-93). The tagged text parser contains over 400 grammar production rules. Some examples of the part of speech tagger identification are given in Figure 8.8.

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

8.8 Part of Speech Tags

The TTP parse trees are header-modifier pairs where the header is the main concept and the modifiers are the additional descriptors that form the concept and eliminate ambiguities. Figure 4.9 gives an example of a regularized parse tree structure generated for the independent clause:

The former Soviet President has been a local hero ever since a Russian tank invaded Wisconsin.

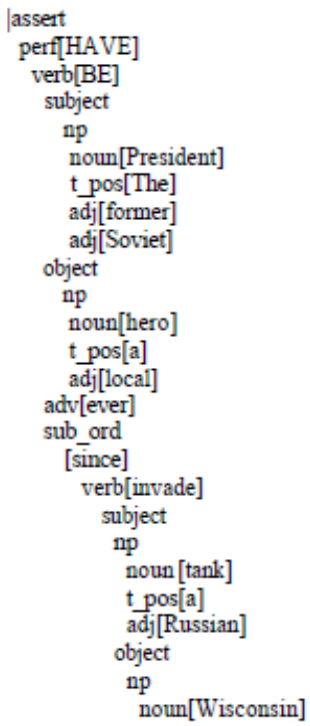


Figure 8.9 TTP Parse Tree

This structure allows for identification of potential term phrases usually based upon noun identification. To determine if a header-modifier pair warrants indexing, Strzalkowski calculates a value for Informational Contribution (IC) for each element in the pair. Higher values of IC indicate a potentially stronger semantic relationship between terms. The basis behind the IC formula is a conditional probability between the terms.

The formula for IC between two terms (x,y) is:

$$IC(x,[x,y]) = \frac{fx,y}{Nx + Dx - 1}$$

where $f_{x,y}$ is the frequency of (x,y) in the database, n_x is the number of pairs in which "x" occurs at the same position as in (x,y) and D(x) is the dispersion parameter which is the

number of distinct words with which x is paired. When IC= 1, x occurs only with $\lambda\,(\xi^{x'\lambda}=u^{x}\ \text{gnq}\ q^{x}\approx 1)$.

Nominal compounds are the source of much inaccurate identification in creating header-modifier pairs. Use of statistical information on frequency of occurrence of phrases can eliminate some combinations that occur infrequently and are not meaningful.

The next challenge is to assign weights to term phrases. The most popular term weighting scheme uses term frequencies and inverse document frequencies with normalization based upon item length to calculate weights assigned to terms (see Section 5.2.2.2). Term phrases have lower frequency occurrences than the individual terms. Using natural language processing, the focus is on semantic relationships versus frequency relationships. Thus weighting schemes such as inverse document frequency require adjustments so that the weights are not overly diminished by the potential lower frequency of the phrases.

For example, the weighting scheme used in the New York University system uses the following formula for weighting phrases:

$$\text{weight}(\text{Phrase}_i) = (C_1 * \log(\text{termf}) + C_2 * \alpha(N,i)) * \text{IDF}$$

where $\alpha(N,i)$ is 1 for i<N and 0 otherwise and C1 and C2 are normalizing factors. The N assumes the phrases are sorted by IDF value and allows the top "N" highest IDF (inverse document frequency) scores to have a greater effect on the overall weight than other terms.

8.3.2 Natural Language Processing

Lexical analysis determining verb tense, plurality and part of speech is assumed to have been completed prior to the following additional processing. Natural language processing not only produces more accurate term phrases, but can provide higher level semantic information identifying relationships between concepts.

The DR-LINK system (Liddy-93) and its commercial implementation via Textwise System adds the functional processes Relationship Concept Detectors, Conceptual Graph Generators and Conceptual Graph Matchers that generate higher level linguistic relationships including semantic and discourse level relationships. This system is representative of natural language based processing systems. During the first phase of this approach, the processing tokens in the document are mapped to Subject Codes as defined by the codes in the Longman's Dictionary of Common English (LDOCE). Disambiguation uses *a priori* statistical term relationships and the ordering of the subject codes in the LDOCE, which indicates most

likely assignment of a term to a code. These codes equate to index term assignment and have some similarities to the concept-based systems discussed in Section 4.4.

The next phase is called the Text Structurer, which attempts to identify general discourse level areas within an item. Thus a news story may be subdivided into areas associated with EVALUATION (opinions), Main event (basic facts), and Expectations (Predictions). These have been updated to include Analytical Information, Cause/Effect Dimension and Attributed Quotations in the more recent versions of DR-LINK (see http://199.100.96.2 on the Internet). These areas can then be assigned higher weighting if the user includes "Preference" in a search statement. The system also attempts to determine TOPIC statement identifiers. Natural language processing is not just determining the topic statement(s) but also assigning semantic attributes to the topic such as time frame (past, present, future). To perform this type analysis, a general model of the predicted text is needed. For example, news items likely follow a model proposed by van Dijk (Dijk-88). Liddy reorganized this structure into a News Schema Components consisting of Circumstance, Consequence, Credentials, Definition, Error, Evaluation, Expectation, History, Lead, Main Event, No Comment, Previous Event, References and Verbal reaction. Each sentence is evaluated and assigned weights associated

with its possible inclusion in the different components. Thus, if a query is oriented toward a future activity, then, in addition to the subject code vector mapping, it would weight higher terms associated with the Expectation component.

The next level of semantic processing is the assignment of terms to components, classifying the intent of the terms in the text and identifying the topical statements. The next level of natural language processing identifies interrelationship between the concepts. For example, there may be two topics within an item "national elections" and "guerrilla warfare." The relationship "as a result of" is critical to link the order of these two concepts. This process clarifies if the elections were caused by the warfare or the warfare caused by the elections.

Significant information is lost by not including the connector relationships. These types of linkages are generated by general linguistic cues (words in text) that are fairly general and domain independent.

The final step is to assign final weights to the established relationships. The relationships are typically envisioned as triples with two concepts and a relationship between them. Although all relationships are possible, constructing a system requires the selection of a subset of possible relationships and the rules to locate the relationships. The weights are based upon a

combination of statistical information and values assigned to the actual words used in establishing the linkages. Passive verbs would receive less weight than active verbs.

The additional information beyond the indexing is kept in additional data structures associated with each item. This information is used whenever it is implicitly included in a search statement that is natural language based or explicitly requested by the user.

8.4 Concept Indexing

Natural language processing starts with a basis of the terms within an item and extends the information kept on an item to phrases and higher level concepts such as the relationships between concepts. In the DR-LINK system, terms within an item are replaced by an associated Subject Code. Use of subject codes or some other controlled vocabulary is one way to map from specific terms to more general terms. Often the controlled vocabulary is defined by an organization to be representative of the concepts they consider important representations of their data. Concept indexing takes the abstraction a level further. Its goal is to gain the implementation advantages of an index term system but use concepts instead of terms as the basis for the index, producing a reduced dimension vector space.

Rather than *a priori* defining a set of concepts that the terms in an item are mapped to, concept indexing can start with a number of unlabeled concept classes and let the information in the items define the concepts classes created. The process of automatic creation of concept classes is similar to the automatic generation of thesaurus classes. The process of mapping from a specific term to a concept that the term represents is complex because a term may represent multiple different concepts to different degrees. A term such as “automobile” could be associated with concepts such as “vehicle,” “transportation,” “mechanical device,” “fuel,” and “environment.” The term “automobile” is strongly related to “vehicle,” lesser to “transportation” and much lesser the other terms. Thus a term in an item needs to be represented by many concept codes with different weights for a particular item.

An example of applying a concept approach is the Convectis System from HNC Software Inc. (Caid-93, Carleton-95). The basis behind the generation of the concept approach is a neural network model (Waltz-85). Context vector representation and its application to textual items are described by Gallant (Gallant- 91a, Gallant-91b). If a vector approach is envisioned, then there are a finite number of concepts that provide coverage over all of the significant concepts required to index a database of items. The goal of the indexing is to allow the user

to find required information, minimizing the reviewing of items that are non-relevant. In an ideal environment there would be enough vectors to account for all possible concepts and thus they would be orthogonal in an "N" dimensional vector-space model. It is difficult to find a set of concepts that are orthogonal with no aspects in common. Additionally, implementation tradeoffs naturally limit the number of concept classes that are practical. These limitations increase the number of classes to which a processing token is mapped. The Convectis system uses neural network algorithms and terms in a similar context (proximity) of other terms as a basis for determining which terms are related and defining a particular concept. A term can have different weights associated with different concepts as described. The definition of a similar context is typically defined by the number of non-stop words separating the terms. The farther apart terms are, the less coupled the terms are associated within a particular concept class. Existing terms already have a mapping to concept classes. New terms can be mapped to existing classes by applying the context rules to the classes that terms near the new term are mapped. Special rules must be applied to create a new concept class. The following example demonstrates how the process would work for the term "automobile."

TERM: automobile

	Wieghts for associated concepts
Vehicle	0.65
Transportation	0.60
Environment	0.35
Fuel	0.33
Mechanical Device	0.15

Vector Representation Automobile: (0.65, ..., 0.60, ..., 0.35..., 0.33, ..., 0.15)

Figure 8.10 Concept Vector for Automobile

Using the concept representation of a particular term, phrases and complete items can be represented as a weighted average of the concept vectors of the terms in them. The algorithms associated with vectors (e.g., inverse document frequency) can be used to perform the merging of concepts.

Another example of this process is Latent Semantic Indexing (LSI). Its assumption is that there is an underlying or "latent" structure represented by interrelationships between words (Deerwester-90, Dempster-77, Dumais-95, Gildea-99, Hofmann-99). The index contains representations of the "latent semantics" of the item. Like Convectis, the large term-document matrix is decomposed into a small set (e.g., 100-300) of orthogonal factors which use linear combinations of the factors (concepts) to approximate the original matrix. Latent Semantic Indexing uses singular-value decomposition to model the associative relationships between terms similar to eigenvector decomposition and factor analysis (see Cullum-85).

Any rectangular matrix can be decomposed into the product of three matrices. Let *X* be a *m*x*n* matrix such that:

$$X = T_0 \bullet S_0 \bullet D_0'$$

where **T0** and **D0** have orthogonal columns and are *m* x *r* and *r* x *n* matrices,**S0** is an *r* x *r* diagonal matrix and *r* is the rank of matrix *X*. This is the singular value decomposition of *X*. The *k* largest singular values of **S0** are kept along with their corresponding columns in **T0** and **D0** matrices, the resulting matrix:

$$\bar{X} = T_n \bullet S_n \bullet D_n'$$

is the unique matrix of rank *k* that is closest in least squares sense to *X*. The matrix $\bar{X}$, containing the first *k* independent linear components of the original *X* represents the major associations with noise eliminated.

If you consider X to be the term-document matrix (e.g., all possible terms being represented by columns and each item being represented by a row), then truncated singular value decomposition can be applied to reduce the dimmensionality caused by all terms to a significantly smaller dimensionality that is an approximation of the original X:

$$X = U \bullet SV \bullet V'$$

where **u1 ... uk** and $\mathbf{v^1}$ **...** $\mathbf{v^k}$ are left and right singular vectors and **sv1 ... svk** are singualr values. A threshold is used against the full SV diagnonal matrix to determine the cutoff on values to be used for query and document representation (i.e., the dimensionality reduction). Hofmann has modified the standard LSI approach using addional formalism via Probabilistic Latent Semantic Analysis (Hofmann-99).

With so much reduction in the number of words, closeness is determined by patterns of word usage versus specific co-locations of terms. This has the effect of a thesaurus in equating many terms to the same concept. Both terms and documents (as collections of terms) can be represented as weighted vectors in the *k* dimensional space. The selection of *k* is critical to the success of this procedure. If *k* is too small, then there is not enough discrimination between vectors and too many false hits are returned on a search. If *k* is too large, the value of Latent Semantic Indexing is lost and the system equates to a standard vector model.

8.5 Hypertext Linkages

Hypertext data structures must be generated manually although user interface tools may simplify the process. Very little research has been done on the information retrieval aspects of hypertext linkages and automatic mechanisms to use the information of item pointers in creating additional search structures. In effect, hypertext linkages are creating an additional information retrieval dimension. Traditional items can be viewed as two dimensional constructs. The text of the items is one dimension representing the information in the items. Imbedded references are a logical second dimension that has had minimal use in information search techniques. The major use of the citations has been in trying to determine the concepts within an item and clustering items (Salton-83). Hypertext, with its linkages to additional electronic items, can be viewed as networking between items that extends the contents. To understand the total subject of an item it is necessary to follow these additional information concept paths. The imbedding of the linkage allows the user to go immediately to the linked item for additional information. The issue is how to use this additional dimension to locate relevant information. The easiest approach is to do nothing and let the user follow these paths to view items. But this is avoiding one of the challenges in information systems on creating techniques to assist the user in finding relevant information.

Looking at the Internet at the current time, the following are the three classes of mechanisms to help find information.

1. Manually generated indexes
2. Automatically generated indexes
3. Web crawlers (intelligent agents)

YAHOO (http://www.yahoo.com) is an example of the first case where information sources (home pages) are indexed manually into a hyperlinked hierarchy. The user can navigate

through the hierarchy by expanding the hyperlink on a particular topic to see the more detailed subtopics. At some point the user starts to see the end items.

LYCOS (http://www.lycos.com) and Altavista (http://www.altavista.digital.com) automatically go out to other Internet sites and return the text at the sites for automatic indexing. Lycos returns home pages from each site for automatic indexing while Altavista indexes all of the text at a site. None of these approaches use the linkages in items to enhance their indexing.

Webcrawlers (e.g., WebCrawler, OpenText, Pathfinder) and intelligent agents (Coriolis Groups' NetSeeker™) are tools that allow a user to define items of interest and they automatically go to various sites on the Internet searching for the desired information. They are better described as a search tool than an indexing tool that *a priori* analyzes items to assist in finding them via a search.

What is needed is an index algorithm for items that looks at the hypertext linkages as an extension of the concepts being presented in the item where the link exists. Some links that are for references to multi-media imbedded objects would not be part of the indexing process. The Universal Reference Locator (URL) hypertext links can map to another item or to a specific location within an item. The current concept is defined by the information within proximity of the location of the link. The concepts in the linked item, or with a stronger weight the concepts in the proximity of the location included in the link, need to be included in the index of the current item. If the current item is discussing the financial state of Louisiana and a hyperlink is included to a discussion on crop damage due to draughts in the southern states, the index should allow for a "hit" on a search statement including "droughts in Louisiana."

One approach is to view the hyperlink as an extension of the text of the item in another dimension. The index values of the hyperlinked item have a reduced weighted value from contiguous text biased by the type of linkage. The weight of processing tokens appears:

$$\mathbf{Weight_{i,j,k,l}} = (\alpha * \mathbf{Weight_{i,j}} + \beta * \mathbf{Weight_{k,l}}) * (\gamma * \mathbf{Link_{i,k}})$$

where **Weighti,j,k,l** is the Weight associated with processing token "j" in item "i" and processing token "l" in item "k" that are related via a hyperlink. **Linki,k** is the weight associated with strength of the link. It could be a one-level link that is weak or strong, or it could be a multilevel transitive link. α, β and γ are weighting/normalization factors. The

values could be stored in an expanded index structure or calculated dynamically if only the hyperlink relationships between items are available.

Taking another perspective, the system could automatically generate hyperlinks between items. Attempts have been made to achieve this capability, but they suffer from working with static versus dynamic growing databases or ignoring the efficiency needed for an operational environment (Allan-95, Furuta-89, Rearick-91). Kellog and Subhas have proposed a new solution based upon document segmentation and clustering (Kellog-96). They link at both the document and document sub-part level using the cover-coefficient based incremental clustering method (C2ICM) to generate links between the document (document sub-parts) pairs for each cluster. (Can-95). The automatic link generation phase is performed in parallel with the clustering phase. Item pairs in the same cluster are candidates for hyperlinking (link-similarity) if they have a similarity above a given threshold. The process is completed in two phases. In the first phase the document seeds and an estimate of the number of clusters is calculated. In the second phase the items are clustered and the links are created. Rather than storing the link information within the item or storing a persistent link ID within the item and the link information externally, they store all of the link information externally. They create HTML items on demand. When analyzing links missed by their algorithm, three common problems were discovered:

1. Misspellings or multiple word representations (e.g., cabinet maker and cabinetmaker)
2. Parser problems with document segmentation caused by punctuation errors (lines were treated as paragraphs and sentences)
3. Problems occurred when the definition of subparts (smaller sentences) of items was attempted.

A significant portion of errors came from parsing rather than algorithmic problems. This technique has maximum effectiveness for referential links which naturally have higher similarity measures.

The statistical, natural language, concept and hyperlink indexing techniques discussed above have their own strengths and weaknesses. Current evaluations from TREC conferences show that to maximize location of relevant items, applying several different algorithms to the same corpus provides the optimum results, but the storage and processing overhead is significant.

8.6 Summary

Automatic indexing is the preprocessing stage allowing search of items in an Information Retrieval System. Its role is critical to the success of searches in finding relevant items. If the concepts within an item are not located and represented in the index during this stage, the item is not found during search. Some techniques allow for the combinations of data at search time to equate to particular concepts (i.e. postcoordination). But if the words are not properly identified at indexing time and placed in the searchable data structure, the system can not combine them to determine the concept at search time. If an inefficient data structure is selected to hold the index, the system does not scale to accommodate large numbers of items.

The steps in the identification of the processing tokens used in the index process, focuses on the specific characteristics of the processing tokens to support the different search techniques.

There are many ways of defining the techniques. All of the techniques have statistical algorithmic properties. But looking at the techniques from a conceptual level, the approaches are classified as statistical, natural language and concept indexing. Hypertext linkages are placed in a separate class because an algorithm to search items that include linkages has to address dependencies between items. Normally the processing for processing tokens is restricted to an item. The next item may use some corpus statistics that changed by previous items, but does not consider a tight coupling between items. In effect, one item may be considered an extension of another, which should effect the concept identification and representation process.

Of all the statistical techniques, an accurate probabilistic technique would have the greatest benefit in the search process. Unfortunately, identification of consistent statistical values used in the probabilistic formulas has proven to be a formidable task. The assumptions that must be made significantly reduce the accuracy of the search process. Vector techniques have very powerful representations and have been shown to be successful. But they lack the flexibility

to represent items that contain many distinct but overlapping concepts. Bayesian techniques are a way to relax some of the constraints inherent in a pure vector approach, allowing dependencies between concepts within the same item to be represented. Most commercial systems do not try to calculate weighted values at index time. It is easier and more flexible to store the basic word data for each item and calculate the statistics at search time. This allows tuning the algorithms without having to re-index the database. It also allows the combination of statistical and traditional Boolean techniques within the same system.

Natural language systems attempt to introduce a higher level of abstraction indexing on top of the statistical processes. Making use of rules associated with language assist in the disambiguation of terms and provides an additional layer of concepts that are not found in purely statistical systems. Use of natural language processing provides the additional data that could focus searches, reducing the retrieval of non-relevant items. The tendency of users to enter short queries may reduce the benefits of this approach. Concept indexing is a statistical technique whose goal is to determine a canonical representation of the concepts. It has been shown to find relevant items that other techniques miss. In its transformation process, some level of precision is lost. The analysis of enhanced recall over potential reduced precision is still under investigation.

8.7 KEYWORDS

Statistical indexing, Vector model, Term frequency, Signal weighting, document frequency, Bayesian model, Index phrase generation, Concept indexing

8.8 QUESTIONS

1. Discuss the process of data flow in information system.
2. Explain statistical indexing with probabilistic weighting
3. Explain the vector model of indexing
4. Describe simple term frequency algorithm.
5. Write a note on inverse document frequency.
6. What is meant by signal weighting? Explain
7. Discuss the problems with weighting schemes.
8. Discuss the problems with vector model.
9. Describe the Bayesian model.
10. Describe the process of index phrase generation.
11. What is meant by concept indexing? Give an example.
12. Write a note on hypertext linkages.

8.9 REFERENCES FOR FURTHER STUDY

Anderson, J. D. & Pérez-Carballo, J. (2001). The nature of indexing: How humans and machines analyze messages and texts for retrieval. Part I: Research, and the nature of human indexing. Information Processing & Management, 37(2), 231-254.

Bar-Hillel, Y. (1960). The present status of automatic translation of languages. Advances in Computers, 1, 91-163.

Chen, H, Yim, T, Fye, D & Schatz, B (1995). Automatic thesaurus generation for an electronic community system. Journal of the American Society for Information Society, 46, 175 – 193.

Ellis, D. (1996). Progress and Problems in Information Retrieval. London: Library Association Publishing.

Kowalski, G. J. & Maybury, M. T. (2000). Information storage and retrieval systems: Theory and implementation. 2nd ed. Norvel, Mass.: Kluwer Academic Publishers. Chapter 6: Document and term clustering, pp. 139-163.

Lancaster, F. W. (1991/1998/2003). Indexing and abstracting in theory and practice. London: Library Association. (1st ed. 1991; 2nd ed. 1998; 3rd. ed. 2003).

Chakraborty, A.R., & Chakrabarti, B. (1983). INDEXING: Principles, process, and products. Calcutta: World Press Private Limited.

Hjorland, B. (2007). Semantics and knowledge organization. Annual Review of Information Science and Technology 41: 367-405.

Oyinloye, A.M. (2000). The Nigerian experience: Indexes, indexers, and indexing. Indexer 22: 78-8

Seth, M.A. (2004). Notes on automatic indexing. Available: http://taxonomist.tripod.com/ indexing/ autoindex.html.

Tulic, M. (2005). Automatic indexing. Available: www.anindexer.com

UNIT 9: DOCUMENT AND TERM CLUSTERING

Structure

9.0 INTRODUCTION TO CLUSTERING

Clustering is the process of grouping the data into classes or clusters, so that objects within a cluster have high similarity in comparison to one another but are very dissimilar to objects in other clusters. Dissimilarities are assessed based on the attribute values describing the objects. Often, distance measures are used. Clustering has its roots in many areas, including data mining, statistics, biology, and machine learning.

Document clustering is an automatic grouping of text documents into clusters so that documents within a cluster have high similarity in comparison to one another, but are dissimilar to documents in other clusters. Unlike document classification (Wang, Zhou, and He, 2001), no labelled documents are provided in clustering; hence, clustering is also known as unsupervised learning. Hierarchical document clustering organizes clusters into a tree or a hierarchy that facilitates browsing. The parent-child relationship among the nodes in the tree can be viewed as a topic-subtopic relationship in a subject hierarchy such as the Yahoo! directory.

A cluster of data objects can be treated collectively as one group and so may be considered as a form of data compression. Although classification is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labeling of a large set of training tuples or patterns, which the classifier uses to model each group. It is often more desirable to proceed in the reverse direction: First partition the set of data into groups based on data similarity (e.g., using clustering), and then assign labels to the relatively small number of groups. Additional advantages of such a clustering-based process are that it is adaptable to changes and helps single out useful features that distinguish different groups.

The concept of clustering has been around as long as there have been libraries. One of the first uses of clustering was an attempt to cluster items discussing the same subject. The goal of the clustering was to assist in the location of information. This eventually lead to indexing schemes used in organization of items in libraries and standards associated with use of electronic indexes. Clustering of words originated with the generation of thesauri. Thesaurus, coming from the Latin word meaning "treasure," is similar to a dictionary in that it stores words. Instead of definitions, it provides the synonyms and antonyms for the words. Its primary purpose is to assist authors in selection of vocabulary. Clustering also allows linkages between clusters to be specified. The term class is frequently used as a synonym for the term cluster. They are used interchangeably in this chapter.

The process of clustering follows the following steps:

a. Define the domain for the clustering effort. If a thesaurus is being created, this equates to determining the scope of the thesaurus such as "medical terms." If document clustering is being performed, it is determination of the set of items to be clustered. This can be a subset of the database or the complete database. Defining the domain for the clustering identifies those objects to be used in the clustering process and reduce the potential for erroneous data that could induce errors in the clustering process.

b. Once the domain is determined, determine the attributes of the objects to be clustered. If a thesaurus is being generated, determine the specific words in the objects to be used in the clustering process. Similarly, if documents are being clustered, the clustering process may focus on specific zones within the items (e.g., Title and abstract only, main body of the item but not the references, etc.) that are

to be used to determine similarity. The objective, as with the first step (a.) is to reduce erroneous associations.

c. Determine the strength of the relationships between the attributes whose co-occurrence in objects suggest those objects should be in the same class. For thesauri this is determining which words are synonyms and the strength of their term relationships. For documents it may be defining a similarity function based upon word co-occurrences that determine the similarity between two items.

d. At this point, the total set of objects and the strengths of the relationships between the objects have been determined. The final step is applying some algorithm to determine the class(s) to which each item will be assigned.

There are guidelines (not hard constraints) on the characteristics of the classes:

a. A well-defined semantic definition should exist for each class. There is a risk that the name assigned to the semantic definition of the class could also be misleading. In some systems numbers are assigned to classes to reduce the misinterpretation that a name attached to each class could have. A clustering of items into a class called "computer" could mislead a user into thinking that it includes items on main memory that may actually reside in another class called "hardware."
b. The size of the classes should be within the same order of magnitude. One of the primary uses of the classes is to expand queries or expand the resultant set of retrieved items. If a particular class contains 90 per cent of the objects, that class is not useful for either purpose. It also places in question the utility of the other classes that are distributed across 10 percent of the remaining objects.
c. Within a class, one object should not dominate the class. For example, assume a thesaurus class called "computer" exists and it contains the objects (words/word phrases) "microprocessor," "286-processor," "386- processor" and "pentium." If the term "microprocessor" is found 85 per cent of the time and the other terms are used 5 per cent each, there is a strong possibility that using "microprocessor" as a synonym for "286- processor" will introduce too many errors. It may be better to place "microprocessor" into its own class.
d. Whether an object can be assigned to multiple classes or just one must be decided at creation time. This is a tradeoff based upon the specificity and partitioning capability of the semantics of the objects. Given the ambiguity of language in general, it is better

to allow an object to be in multiple classes rather than constrained to one. This added flexibility comes at a cost of additional complexity in creating and maintaining the classes.

There are additional important decisions associated with the generation of thesauri that are not part of item clustering (Aitchison-72):

a. **Word coordination approach**: specifies if phrases as well as individual terms are to be clustered.
b. **Word relationships**: when the generation of a thesaurus includes a human interface (versus being totally automated), a variety of relationships between words are possible. Aitchison and Gilchrist (Aitchison-72) specified three types of relationships: equivalence, hierarchical and non-hierarchical. Equivalence relationships are the most common and represent synonyms. The definition of a synonym allows for some discretion in the thesaurus creation, allowing for terms that have significant overlap but differences. Thus the terms photograph and print may be defined as synonyms even though prints also include lithography. The definition can even be expanded to include words that have the same "role" but not necessarily the same meaning. Thus the words "genius" and "moron" may be synonyms in a class called "intellectual capability." A very common technique is hierarchical relationships where the class name is a general term and the entries are specific examples of the general term. The previous example of "computer" class name and "microprocessor," "pentium," etc. is an example of this case. Nonhierarchical relationships cover other types of relationships such as "object"-"attribute" that would contain "employee" and "job title."

   A more recent word relationship scheme (Wang-85) classified relationships as Parts-Wholes, Collocation, Paradigmatic, Taxonomy and Synonymy, and Antonymy. The only two of these classes that require further amplification are collocation and paradigmatic. Collocation is a statistical measure that relates words that co-occur in the same proximity (sentence, phrase, paragraph). Paradigmatic relates words with the same semantic base such as "formula" and "equation."

   In the expansion to semantic networks other relationships are included such as contrasted words, child-of (sphere is a child-of geometric volume), parent-of, part-of (foundation is part of a building), and contains part-of (bicycle contains parts-of wheel, handlebars) (RetrievalWare-95).

c. **Homograph resolution**: a homograph is a word that has multiple, completely different meanings. For example, the term "field" could mean an electronic field, a field of grass, etc. It is difficult to eliminate homographs by supplying a unique meaning for every homograph (limiting the thesaurus domain helps). Typically the system allows for homographs and requires that the user interact with the system to select the desired meaning. It is possible to determine the correct meaning of the homograph when a user enters multiple search terms by analyzing the other terms entered (hay, crops, and field suggest the agricultural meaning for field).

d. **Vocabulary constraints:** this includes guidelines on the normalization and specificity of the vocabulary. Normalization may constrain the thesaurus to stems versus complete words. Specificity may eliminate specific words or use general terms for class identifiers.

As is evident in these guidelines, clustering is as much an arcane art as it is a science. Good clustering of terms or items assists the user by improving recall. But typically an increase in recall has an associated decrease in precision. Automatic clustering has the imprecision of information retrieval algorithms, compounding the natural ambiguities that come from language. Care must be taken to ensure that the increases in recall are not associated with such decreases in precision as to make the human processing (reading) of the retrieved items unmanageable. The key to successful clustering lies in selection of a good measure of similarity and selection of a good algorithm for placing items in the same class. When hierarchical item clustering is used, there is a possibility of a decrease in recall. The only solution to this problem is to make minimal use of the hierarchy.

9.1 THESAURUS GENERATION

Thesauri are valuable structures for Information Retrieval systems. A thesaurus provides a precise and controlled vocabulary which serves to coordinate document indexing and document retrieval. In both indexing and retrieval, a thesaurus may be used to select the most appropriate terms. Additionally, the thesaurus can assist the searcher in reformulating search strategies if required.

Manual generation of clusters usually focuses on generating a thesaurus (i.e., clustering terms versus items) and has been used for hundreds of years. As items became available in electronic form, automated term statistical clustering techniques became available. Automatically generated thesauri contain classes that reflect the use of words in the corpora.

The classes do not naturally have a name, but are just groups of statistically similar terms. The optimum technique for generating the classes requires intensive computation. Other techniques starting with existing clusters can reduce the computations required but may not produce optimum classes.

There are three basic methods for generation of a thesaurus; hand crafted, co-occurrence, and header-modifier based. Using manually made thesauri only helps in query expansion if the thesauri are domain specific for the domain being searched. General thesaurus (e.g., WordNet) does not help as much because of the many different meanings for the same word (Voorhees-93, Voorhees-94). Techniques for co-occurrence creation of thesauri are described in detail below. In header-modifier based thesauri term relationships are found based upon linguistic relationships. Words appearing in similar grammatical contexts are assumed to be similar (Hindle-90, Grafenstette-94, Jing-94, Ruge-92). The linguistic parsing of the document discovers the following syntactical structures: Subject-Verb, Verb-Object, Adjective-Noun, and Noun-Noun. Each noun has a set of verbs, adjectives and nouns that it co-occurs with, and a mutual information value is calculated for each using typically a log function. Then a final similarity between words is calculated using the mutual information to classify the terms.

The differences between manually and automatically generated thesauri for the field of IR. The following tables illustrate those in the fields of structure, goal, construction and verification.

	Manual	**Automatic**
Structure	Hierarchy of thesaurus terms High level of coordination Many types of relations between terms Complex normalization rules Field limits are specified by the creators	Many different approaches, but not always hierarchical Lower level of coordination (phrase selection not easy to do) Simple normalization rules; hard to separate homographs. Field limits are specified by the collection
Goal	Main goal is to precisely define the vocabulary to be used in a technical field Due to this precise definition, useful to index documents. Assistance in developing search strategy Assistance in retrieval through query expansion/contraction	Depending on level of coordination, can be used for indexing. Main use is to assist in retrieval through (possibly automated) query expansion/contraction

Construction	define boundaries of field, subdivide into areas fix characteristics collect term definitions from a variety of sources (including encyclopaedias, expert advice, …) Analyze data and set up relationships. From these, a hierarchy should arise Evaluate consistency, incorporate new terms or change relationships [3, 4] Create an inverted form, and release the thesaurus Periodical updates	Identify the collection to be used Fix characteristics (less degrees of liberty here) Select and normalize terms, phrase construction. Statistical analysis to find relationships (only one kind) If desired, organize as a hierarchy
Verification	Soundness and coverage of concept classification	Ability to improve retrieval performance

9.1.1 Manual Clustering

The manual clustering process follows the generation of a thesaurus. The first step is to determine the domain for the clustering. Defining the domain assists in reducing ambiguities caused by homographs and helps focus the creator. Usually existing thesauri, concordances from items that cover the domain and dictionaries are used as starting points for generating the set of potential words to be included in the new thesaurus. A concordance is an alphabetical listing of words from a set of items along with their frequency of occurrence and references of which items in which they are found.

The art of manual thesaurus construction resides in the selection of the set of words to be included. Care is taken to not include words that are unrelated to the domain of the thesaurus or those that have very high frequency of occurrence and thus hold no information value (e.g., the term Computer in a thesaurus focused on data processing machines). If a concordance is used, other tools such as KWOC, KWIC or KWAC may help in determining useful words. A Key Word Out of Context (KWOC) is another name for a concordance. Key Word In Context (KWIC) displays a possible term in its phrase context. It is structured to identify easily the location of the term under consideration in the sentence. Key Word And Context (KWAC) displays the keywords followed by their context. Figure 1.1 shows the various displays for “computer design contains memory chips” (NOTE: the phrase is assumed to be from doc4; the other frequency and document ids for KWOC were created for this example.) In the Figure 1.1 the character “/” is used in KWIC to indicate the end of the

phrase. The KWIC and KWAC are useful in determining the meaning of homographs. The term "chips" could be wood chips or memory chips. In both the KWIC and KWAC displays, the editor of the thesaurus can read the sentence fragment associated with the term and determine its meaning. The KWOC does not present any information that would help in resolving this ambiguity.

KWOC

TERM	FREQ	ITEM Ids
chips	2	doc2, doc4
computer	3	doc1, doc4, doc10
design	1	doc4
memory	3	doc3, doc4, doc8, doc12

KWIC

chips/	computer design contains memory
computer	design contains memory chips/
design	contains memory chips/ computer
memory	chips/ computer design contains

KWAC

chips	computer design contains memory chips
computer	computer design contains memory chips
design	computer design contains memory chips
memory	computer design contains memory chips

Figure 9.1 Example of KWOC, KWIC and KWAC

Once the terms are selected they are clustered based upon the word relationship guidelines and the interpretation of the strength of the relationship. This is also part of the art of manual creation of the thesaurus, using the judgment of the human analyst. The resultant thesaurus undergoes many quality assurance reviews by additional editors using some of the guidelines already suggested before it is finalized.

9.1.2 Automatic Term Clustering

There are many techniques for the automatic generation of term clusters to create statistical thesauri. They all use as their basis the concept that the more frequently two terms co-occur in the same items, the more likely they are about the same concept. They differ by the completeness with which terms are correlated. The more complete the correlation, the higher the time and computational overhead to create the clusters. The most complete process computes the strength of the relationships between all combinations of the "n" unique words with an overhead of $o(n^2)$ Other techniques start with an arbitrary set of clusters and iterate on the assignment of terms to these clusters. The simplest case employs one pass of the data in

creation of the clusters. When the number of clusters created is very large, the initial clusters may be used as a starting point to generate more abstract clusters creating a hierarchy.

The steps described in Section 1.1 apply to the automatic generation of thesauri. The basis for automatic generation of a thesaurus is a set of items that represents the vocabulary to be included in the thesaurus. Selection of this set of items is the first step of determining the domain for the thesaurus. The processing tokens (words) in the set of items are the attributes to be used to create the clusters. Implementation of the other steps differs based upon the algorithms being applied. In the following sections a term is usually restricted to be included in only one class. It is also possible to use a threshold instead of choosing the highest value, allowing a term to be assigned to all of the classes that it could be included in above the threshold. The automated method of clustering documents is based upon the polythetic clustering (Van Rijsbergen-79) where each cluster is defined by a set of words and phrases. Inclusion of an item in a cluster is based upon the similarity of the item's words and phrases to those of other items in the cluster.

9.1.2.1 Complete Term Relation Method

In the complete term relation method, the similarity between every term pair is calculated as a basis for determining the clusters. The easiest way to understand this approach is to consider the vector model. The vector model is represented by a matrix where the rows are individual items and the columns are the unique words (processing tokens) in the items. The values in the matrix represent how strongly that particular word represents concepts in the item. Figure 9.2 provides an example of a database with 5 items and 8 terms.

To determine the relationship between terms, a similarity measure is required. The measure calculates the similarity between two terms.

	Term1	Term2	Term3	Term4	Term5	Term6	Term7	Term8
Item 1	0	4	0	0	0	2	1	3
Item 2	3	1	4	3	1	2	0	1
Item 3	3	0	0	0	3	0	3	0
Item 4	0	1	0	3	0	0	2	0
Item 5	2	2	2	3	1	4	0	2

Figure 9.2 Vector Example

The similarity measure is not critical in understanding the methodology so the following simple measure is used:

$$SIM(Term_i, Term_j) = \Sigma\ (Term_{k,i})\ (Term_{k,j})$$

where “k” is summed across the set of all items. In effect the formula takes the two columns of the two terms being analyzed, multiplying and accumulating the values in each row. The results can be paced in a resultant “m” by “m” matrix, called a Term-Term Matrix (Salton-83), where “m” is the number of columns (terms) in the original matrix. This simple formula is reflexive so that the matrix that is generated is symmetric. Other similarity formulas could produce a non-symmetric matrix. Using the data in Figure 1.2, the Term-Term matrix produced is shown in Figure 1.3. There are no values on the diagonal since that represents the autocorrelation of a word to itself. The next step is to select a threshold that determines if two terms are considered similar enough to each other to be in the same class. In this example, the threshold value of 10 is used. Thus two terms are considered similar if the similarity value between them is 10 or greater. This produces a new binary matrix called the Term Relationship matrix (Figure 1.4) that defines which terms are similar. A one in the matrix indicates that the terms specified by the column and the row are similar enough to be in the same class. Term 7 demonstrates that a term may exist on its own with no other similar terms identified. In any of the clustering processes described below this term will always migrate to a class by itself.

The final step in creating clusters is to determine when two objects (words) are in the same cluster. There are many different algorithms available. The following algorithms are the most common: cliques, single link, stars and connected components.

	Term1	Term2	Term3	Term4	Term5	Term6	Term7	Term8
Term 1		7	16	15	14	14	9	7
Term 2	7		8	12	3	18	6	17
Term 3	16	8		18	6	16	0	8
Term 4	15	12	18		6	18	6	9
Term 5	14	3	6	6		6	9	3
Term 6	14	18	16	18	6		2	16
Term 7	9	6	0	6	9	2		3
Term 8	7	17	8	9	3	16	3	

Figure 9.3 Term-Term Matrix

	Term1	Term2	Term3	Term4	Term5	Term6	Term7	Term8
Term 1		0	1	1	1	1	0	0
Term 2	0		0	1	0	1	0	1
Term 3	1	0		1	0	1	0	0
Term 4	1	1	1		0	1	0	0
Term 5	1	0	0	0		0	0	0
Term 6	1	1	1	1	0		0	1
Term 7	0	0	0	0	0	0		0
Term 8	0	1	0	0	0	1	0	

Figure 9.4 Term Relationship Matrix

Cliques require all items in a cluster to be within the threshold of all other items. The methodology to create the clusters using cliques is:

0. Let i = 1
1. Select termi and place it in a new class
2. Start with termk where r = k = i + 1
3. Validate termk if is within the threshold of all terms within the current class
4. If not, let k = k + 1
5. If k > m (number of words) then r = r + 1
   if r = m then go to 6 else
   k = r
   create a new class with in it
   go to 3
   else
   go to 3
6. If current class only has termi in it and there are other classes with termi in them then delete current class
   else
   i = i + 1
7. If i = m + 1 then
   go to 8
   else
   go to 1
8. Eliminate any classes that duplicate or are subsets of other classes.

Applying the algorithm to Figure 1.4, the following classes are created:

Class 1 (Term 1, Term 3, Term 4, Term 6)

Class 2 (Term 1, Term 5)

Class 3 (Term 2, Term 4, Term 6)

Class 4 (Term 2, Term 6, Term 8)

Class 5 (Term 7)

Notice that Term 1 and Term 6 are in more than one class. A characteristic of this approach is that terms can be found in multiple classes.

In single link clustering the strong constraint that every term in a class is similar to every other term is relaxed. The rule to generate single link clusters is that any term that is similar to any term in the cluster can be added to the cluster. It is impossible for a term to be in two different clusters. This in effect partitions the set of terms into the clusters. The algorithm is:

1. Select a term that is not in a class and place it in a new class

2. Place in that class all other terms that are related to it
3. For each term entered into the class, perform step 2
4. When no new terms can be identified in step 2, go to step 1.

Applying the algorithm for creating clusters using single link to the Term Relationship Matrix, Figure 1.4, the following classes are created:

Class 1 (Term 1, Term 3, Term 4, Term 5, Term 6, Term 2, Term 8)

Class 2 (Term 7)

There are many other conditions that can be placed on the selection of terms to be clustered. The Star technique selects a term and then places in the class all terms that are related to that term (i.e., in effect a star with the selected term as the core). Terms not yet in classes are selected as new seeds until all terms are assigned to a class. There are many different classes that can be created using the Star technique. If we always choose as the starting point for a class the lowest numbered term not already in a class, using Figure 1.4, the following classes are created:

Class 1 (Term 1, Term 3, Term 4, Term 5, Term 6)

Class 2 (Term 2, Term 4, Term 8, Term 6)

Class 3 (Term 7)

This technique allows terms to be in multiple clusters (e.g., Term 4). This could be eliminated by expanding the constraints to exclude any term that has already been selected for a previous cluster

The String technique starts with a term and includes in the class one additional term that is similar to the term selected and not already in a class. The new term is then used as the new node and the process is repeated until no new terms can be added because the term being analyzed does not have another term related to it or the terms related to it are already in the class. A new class is started with any term not currently in any existing class. Using the additional guidelines to select the lowest number term similar to the current term and not to select any term already in an existing class produces the following classes:

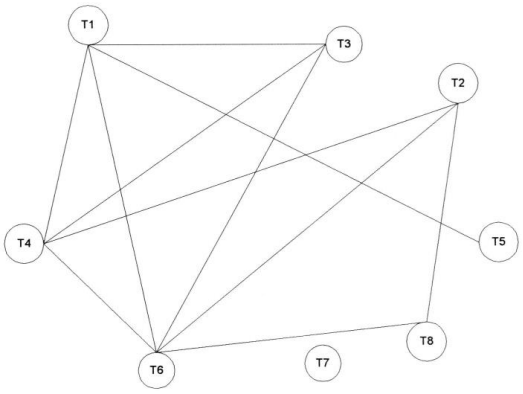


Figure 9.5 Network Diagram of Term Similarities

Class 1 (Term 1, Term 3, Term 4, Term 2, Term 6, Term 8)

Class 2 (Term 5)

Class 3 (Term 7)

A technique to understand these different algorithms for generating classes is based upon a network diagram of the terms. Each term is considered a node and arcs between the nodes indicate terms that are similar. A network diagram for Figure 1.4 is given in Figure 1.5. To determine cliques, sub-networks are identified where all of the items are connected by arcs. From this diagram it is obvious that Term 7 (T7) is in a class by itself and Term 5 (T5) is in a class with Term 1 (T1). Other common structures to look for are triangles and four sided polygons with diagonals. To find all classes for an item, it is necessary to find all subnetworks, where each subnetwork has the maximum number of nodes, that the term is contained. For Term 1 (T1), it is the subnetwork T1, T3, T4, and T6. Term 2 (T2) has two subnetworks: T2, T4, T6 and the subnetwork T2, T6, T8. The network diagram provides a simple visual tool when there are a small number of nodes to identify classes using any of the other techniques.

The clique technique produces classes that have the strongest relationships between all of the words in the class. This suggests that the class is more likely to be describing a particular concept. The clique algorithm produces more classes than the other techniques because the requirement for all terms to be similar to all other terms will reduce the number of terms in a class. This will require more classes to include all the terms. The single link technique partitions the terms into classes. It produces the fewest number of classes and the weakest relationship between terms (Salton-72, Jones-71, Salton-75). It is possible using the single link algorithm that two terms that have a similarity value of zero will be in the same class.

Classes will not be associated with a concept but cover a diversity of concepts. The other techniques lie between these two extremes.

The selection of the technique is also governed by the density of the term relationship matrix and objectives of the thesaurus. When the Term Relationship Matrix is sparse (i.e., contains a few number of ones), then the constraint dependencies between terms need to be relaxed such as in single link to create classes with a reasonable number of items. If the matrix is dense (i.e., lots of ones implying relationships between many terms), then the tighter constraints of the clique are needed so the number of items in a class does not become too large.

Cliques provide the highest precision when the statistical thesaurus is used for query term expansion. The single link algorithm maximizes recall but can cause selection of many non-relevant items. The single link assignment process has the least overhead in assignment of terms to classes, requiring o(n^2) comparisons (Croft-77)

9.1.2.2 Clustering Using Existing Clusters

An alternative methodology for creating clusters is to start with a set of existing clusters. This methodology reduces the number of similarity calculations required to determine the clusters. The initial assignment of terms to the clusters is revised by revalidating every term assignment to a cluster. The process stops when minimal movement between clusters is detected. To minimize calculations, centroids are calculated for each cluster. A centroid is viewed in Physics as the center of mass of a set of objects. In the context of vectors, it will equate to the average of all of the vectors in a cluster.

One way to understand this process is to view the centroids of the clusters as another point in the N-dimensional space where N is the number of items. The first assignment of terms to clusters produces centroids that are not related to the final clustering of terms. The similarity between all existing terms and the centroids of the clusters can be calculated. The term is reallocated to the cluster(s) that has the highest similarity. This process is iterated until it stabilizes.

Calculations using this process are of the order O(n). The initial assignment of terms to clusters is not critical in that the iterative process changes the assignment of terms to clusters.

A graphical representation of terms and centroids illustrates how the classes move after the initial assignment.

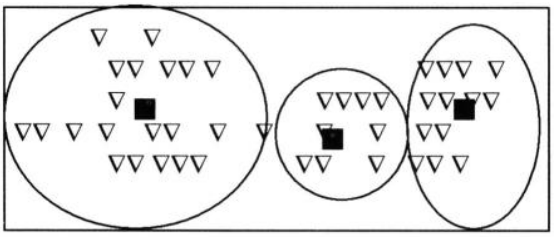


Figure 9.6a Centroids after Reassigning Terms

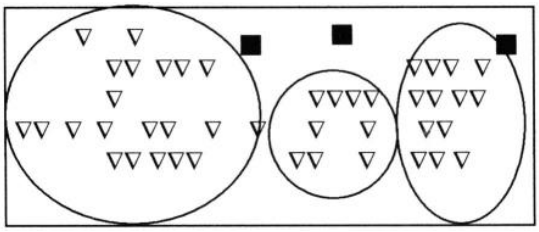


Figure 9.6b. Initial Centroids for Clusters

The solid black box represents the centroid for each of the classes. In Figure 9.6b. the centroids for the first three arbitrary class are shown. The ovals in Figure 9.6b show the ideal cluster assignments for each term. During the next iteration the similarity between every term and the clusters is performed reassigning terms as needed. The resulting new centroid for the new clusters is again shown as black squares in Figure 9.6a. The new centroids are not yet perfectly associated with the ideal clusters, but they are much closer. The process continues until it stabilizes.

The following example of this technique uses Figure 1.2 as our weighted environment, and assumes we arbitrarily placed Class 1 = (Term 1 and Term 2), Class 2 = (Term3 and Term 4) and Class 3 = (Term5 and Term 6). This would produce the following centroids for each class:

Class 1 = (0 +4)/2, (3 + 1)/2, (3 + 0)/2, (0 +1)/2, (2 + 2)/2
= 4/2, 4/2, 3/2, 1/2, 4/2

Class 2 = 0/2, 7/2, 0/2, 3/2, 5/2

Class 3 = 2/2 3/2, 3/2, 0/2, 5/2

Each value in the centroid is the average of the weights of the terms in the cluster for each item in the database. For example in Class 1 the first value is calculated by averaging the weights of Term 1 and Term 2 in Item 1. For Class 2 and 3 the numerator is already the sum of the weights of each term. For the next step, calculating similarity values, it is often easier to leave the values in fraction form.

Applying the simple similarity measure defined in Section 6.2.2.1 between each of the 8 terms and 3 centroids just calculated comes up with the following assignment of similarity weights and new assignment of terms to classes in the row Assign shown in Figure 1.7:

	Term1	Term2	Term3	Term4	Term5	Term6	Term7	Term8
Class 1	29/2	29/2	24/2	27/2	17/2	32/2	15/2	24/2
Class 2	31/2	20/2	38/2	45/2	12/2	34/2	6/2	17/2
Class 3	28/2	21/2	22/2	24/2	17/2	30/2	11/2	19/2
Assign	Class2	Class1	Class2	Class2	Class3	Class2	Class1	Class1

Figure 9.7 Iterated Class Assignments

In the case of Term 5, where there is tie for the highest similarity, either class could be assigned. One technique for breaking ties is to look at the similarity weights of the other items in the class and assign it to the class that has the most similar weights. The majority of terms in Class 1 have weights in the high 20's/2, thus Term 5 was assigned to Class 3. Term 7 is assigned to Class 1 even though its similarity weights are not in alignment with the other terms in that class. Figure 1.8 shows the new centroids and results of similarity comparisons for the next iteration.

Class 1 = 8/3, 2/3, 3/3, 3/3, 4/3

Class 2 = 2/4, 12/4, 3/4, 3/4, 11/4

Class 3 = 0/1, 1/1, 3/1, 0/1, 1/1

	Term1	Term2	Term3	Term4	Term5	Term6	Term7	Term8
Class 1	23/3	45/3	16/3	27/3	15/3	36/3	23/3	34/3
Class 2	67/4	45/4	70/4	78/4	33/4	72/4	17/4	40/4
Class 3	12/1	3/1	6/1	6/1	11/1	6/1	9/1	3/1
Assign	Class2	Class1	Class2	Class2	Class3	Class2	Class3	Class1

Figure 9.8 New Centroids and Cluster Assignments

In this iteration of the process,, the only change is Term 7 moves from Class 1 to Class 3. This is reasonable, given it was not that strongly related to the other terms in Class 1.

Although the process requires fewer calculations than the complete term relationship method, it has inherent limitations. The primary problem is that the number of classes is defined at the start of the process and cannot grow. It is possible for there to be fewer classes at the end of the process. Since all terms must be assigned to a class, it forces terms to be allocated to classes, even if their similarity to the class is very weak compared to other terms assigned.

9.1.2.3 One Pass Assignments

This technique has the minimum overhead in that only one pass of all of the terms is used to assign terms to classes. The first term is assigned to the first class. Each additional term is compared to the centroids of the existing classes. A threshold is chosen. If the item is greater than the threshold, it is assigned to the class with the highest similarity. A new centroid has to be calculated for the modified class. If the similarity to all of the existing centroids is less than the threshold, the term is the first item in a new class. This process continues until all items are assigned to classes. Using the system defined in Figure 1.3, with a threshold of 10 the following classes would be generated:

Class 1 = Term 1, Term 3, Term 4

Class 2 = Term 2, Term 6, Term 8

Class 3 = Term 5

Class 4 = Term 7

NOTE: the centroid values used during the one-pass process:

Class1 (Term1, Term3) = 0, 7/2, 3/2, 0, 4/2

Class1 (Term1, Term3, Term4) = 0, 10/3, 3/3, 3/3, 7/3

Class2 (Term2, Term6) = 6/2, 3/2, 0/2, 1/2, 6/2

Although this process has minimal computation on the order of O(n), it does not produce optimum clustered classes. The different classes can be produced if the order in which the items are analyzed changes. Items that would have been in the same cluster could appear in different clusters due to the averaging nature of centroids.

9.2 ITEM CLUSTERING

Clustering of items is very similar to term clustering for the generation of thesauri. Manual item clustering is inherent in any library or filing system. In this case someone reads the item and determines the category or categories to which it belongs. When physical clustering occurs, each item is usually assigned to one category. With the advent of indexing, an item is physically stored in a primary category, but it can be found in other categories as defined by the index terms assigned to the item.

With the advent of electronic holdings of items, it is possible to perform automatic clustering of the items. The techniques described for the clustering of terms in Sections 1.2.2.1 through 1.2.2.3 also apply to item clustering. Similarity between documents is based upon two items that have terms in common versus terms with items in common. Thus, the similarity function is performed between rows of the item matrix. Using Figure 1.2 as the set of items and their terms and similarity equation:

$$SIM(Item_i, Item_j) = \Sigma\ (Term_{i,k})\ (Term_{j,k})$$

as *k* goes from 1 to 8 for the eight terms, an Item-Item matrix is created (Figure 1.9). Using a threshold of 10 produces the Item Relationship matrix shown in Figure 1.10.

	Item 1	Item 2	Item 3	Item 4	Item 5
Item 1		11	3	6	22
Item 2	11		12	10	36
Item 3	3	12		6	9
Item 4	6	10	6		11
Item 5	22	36	9	11	

Figure 9.9 Item/Item Matrix

	Item1	Item2	Item3	Item4	Item5
Item1		1	0	0	1
Item2	1		1	1	1
Item3	0	1		0	0
Item4	0	1	0		1
Item5	1	1	0	1	

Figure 9.10 Item Relationship Matrix

Using the Clique algorithm for assigning items to classes produces the following classes based upon Figure 9.10:

Class 1 = Item 1, Item 2, Item 5

Class 2 = Item 2, Item 3

Class 3 = Item 2, Item 4, Item 5

Application of the single link technique produces:

Class 1 = Item 1, Item 2, Item 5, Item 3, Item 4

All the items are in this one cluster, with Item 3 and Item 4 added because of their similarity to Item 2. The Star technique (i.e., always selecting the lowest non-assigned item) produces:

Class 1 - Item 1, Item 2, Item 5

Class 2 - Item 3, Item 2

Class 3 - Item4, Item2, Item5

Using the String technique and stopping when all items are assigned to classes produces the following:

Class 1 - Item 1, Item 2, Item 3

Class 2 - Item 4, Item 5

In the vocabulary domain homographs introduce ambiguities and erroneous hits. In the item domain multiple topics in an item may cause similar problems. This is especially true when the decision is made to partition the document space. Without pre-coordination of semantic concepts, an item that discusses “Politics” in “America” and “Economics” in “Mexico” could get clustered with a class that is focused around “Politics” in “Mexico.”

Clustering by starting with existing clusters can be performed in a manner similar to the term model. Let’s start with item 1 and item 3 in Class 1, and item 2 and item 4 in Class 2. The centroids are:

Class 1 = 3/2, 4/2, 0/2, 0/2, 3/2, 2/2, 4/2, 3/2

Class 2 = 3/2, 2/2, 4/2, 6/2, 1/2, 2/2, 2/2, 1/2

The results of recalculating the similarities of each item to each centroid and reassigning terms is shown in Figure 9.11.

	Class1	Class 2	Assign
Item 1	33/2	17/2	Class 1
Item 2	23/2	51/2	Class 2
Item 3	30/2	18/2	Class 2
Item 4	8/2	24/2	Class 2
Item 5	31/2	47/2	Class 2

Figure 9.11 Item Clustering with Initial Clusters

Finding the centroid for Class 2, which now contains four items, and recalculating the similarities does not result in reassignment for any of the items. Instead of using words as a basis for clustering items, the Acquaintance system uses n-grams (Damashek-95, Cohen-95). Not only does their algorithm cluster items, but when items can be from more than one language, it will also recognize the different languages.

9.3 HIERARCHY OF CLUSTERS

Hierarchical clustering in Information Retrieval focuses on the area of hierarchical agglomerative clustering methods (HACM) (Willet-88). The term agglomerative means the clustering process starts with un-clustered items and performs pair wise similarity measures to determine the clusters. Divisive is the term applied to starting with a cluster and breaking it down into smaller clusters. The objectives of creating a hierarchy of clusters are to:

a. Reduce the overhead of search
b. Provide for a visual representation of the information space
c. Expand the retrieval of relevant items.

Search overhead is reduced by performing top-down searches of the centroids of the clusters in the hierarchy and trimming those branches that are not relevant. It is difficult to create a visual display of the total item space. Use of dendograms along with visual cues on the size of clusters (e.g., size of the ellipse) and strengths of the linkages between clusters (e.g., dashed lines indicate reduced similarities) allows a user to determine alternate paths of browsing the database (see Figure 1.12). The dendogram allows the user to determine which clusters to be reviewed are likely to have items of interest. Even without the visual display of the hierarchy, a user can use the logical hierarchy to browse items of interest.

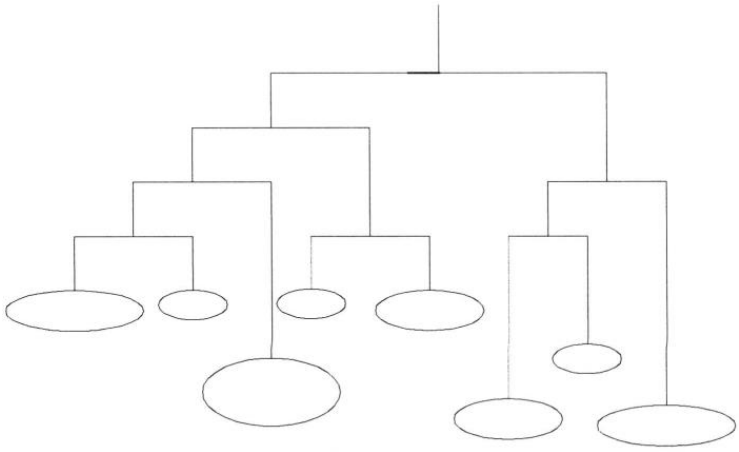


Figure 9.12 Dendogram

A user, once having identified an item of interest, can request to see other items in the cluster. The user can increase the specificity of items by going to children clusters or by increasing the generality of items being reviewed by going to a parent cluster.

Most of the existing HACM approaches can be defined in terms of the Lance-Williams dissimilarity update formula (Lance-66). It defines a general formula for calculating the dissimilarity D between any existing cluster Ck new cluster Cij created by combining clusters Ci and Cj

$$D(C_{i,j},C_k) = \alpha_i D(C_i,C_k) + \alpha_j D(C_j,C_k) + \beta D(C_i,C_j) + \gamma | D(C_i,C_k) - D(C_j,C_k)|$$

By proper selection of α, β, and γ, and the current techniques for HACM can be represented (Frakes-92). In comparing the various methods of creating hierarchical clusters Voorhees and later El-Hamdouchi and Willet determined that the group average method produced the best results on document collections (Voorhees-86, El-Hamdouchi-89).

The similarity between two clusters can be treated as the similarity between all objects in one cluster and all objects in the other cluster. Voorhees showed that the similarity between a cluster centroid and any item is equal to the mean similarity between the item and all items in the cluster. Since the centroid is the average of all items in the cluster, this means that similarities between centroids can be used to calculate the similarities between clusters.

Ward's Method (Ward-63) chooses the minimum square Euclidean distance between points (e.g., centroids in this case) normalized by the number of objects in each cluster. He uses the formula for the variance I, choosing the minimum variance:

$$I_{i,j} = ((m_i m_j)/(m_i + m_j))d_{i,j}^2$$

$$d_{i,j}^2 = \Sigma_{k=1} (x_{i,k} - x_{j,k})^2$$

where **mi** is the number of objects in **Classi** and $\mathbf{dij}^2$ is the square Euclidian distance. The process of selection of centroids can be improved by using the reciprocal nearest neighbour algorithm (Murtaugh-83, Murtaugh-85).

The techniques described in Section 1.2 created independent sets of classes. The automatic clustering techniques can also be used to create a hierarchy of objects (items or terms). The automatic approach has been applied to creating item hierarchies more than in hierarchical statistical thesaurus generation. In the manual creation of thesauri, network relationships are frequently allowed between terms and classes creating an expanded thesaurus called semantic

networks (e.g., in TOPIC and RetrievalWare). Hierarchies have also been created going from general categories to more specific classes of terms. The human creator ensures that the generalization or specification as the hierarchy is created makes semantic sense. Automatic creation of a hierarchy for a statistical thesaurus introduces too many errors to be productive.

But for item hierarchies the algorithms can also be applied. Centroids were used to reduce computation required for adjustments in term assignments to classes. For both terms and items, the centroid has the same structure as any of the items or terms when viewed as a vector from the Item/Term matrix (see Figure 1.2). A term is a vector composed of a column whereas an item is a vector composed of a row. The Scatter/Gather system (Hearst-96) is an example of this technique. In the Scatter/Gather system an initial set of clusters was generated. Each of these clusters was re-clustered to produce a second level. This process iterated until individual items were left at the lowest level.

When the creation of the classes is complete, a centroid can be calculated for each class. When there are a large number of classes, the next higher level in the hierarchy can be created by using the same algorithms used in the initial clustering to cluster the centroids. The only change required may be in the thresholds used. When this process is complete, if there are still too many of these higher level clusters, an additional iteration of clustering can be applied to their centroids. This process will continue until the desired number of clusters at the highest level is achieved.

A cluster can be represented by a category if the clusters were monolithic (membership is based upon a specific attribute). If the cluster is polythetic, generated by allowing for multiple attributes (e.g., words/concepts), then it can best be represented by using a list of the most significant words in the cluster. An alternative is to show a two or three-dimensional space where the clusters are represented by clusters of points. Monolithic clusters have two advantages over polythetic (Sanderson-99): how easy it is for a user to understand the topic of the cluster and the confidence that every item within the cluster will have a significant focus on the topic. For example, YAHOO is a good example of a monolithic cluster environment.

Sanderson and Croft proposed the following methodology to building a concept hierarchy. Rather than just focusing the construction of the hierarchy, they looked at ways of extracting terms from the documents to represent the hierarchy. The terms had the following characteristics:

a. Terms had to best reflect the topics
b. A parent term would refer to a more general concept then its child
c. A child would cover a related subtopic of the parent
d. A directed acyclic graph would represent relationships versus a pure hierarchy.
e. Ambiguous terms would have separate entries in the hierarchy for each meaning.

As a concept hierarchy, it should be represented similar to WordNet (Miller-95) which uses synonyms, antonyms, hyponym / hypernym (is-a/is-a-type-of), and meronym / holonym (has-part/is-a-part-of). Some techniques for generating hierarchies are Grefenstette's use of the similarity of contexts for locating synonyms (Grefenstette-94), use of key phrases (e.g., "such as", "and other") as an indicator of hyponym / hypernym relationships (Hearst-98), use of head and modifier noun and verb phrases to determine hierarchies (Woods-97) and use of a cohesion statistic to measure the degree of association between terms (Forsyth-86). Sanderson and Croft used a test based upon subsumption. It is defined given two terms X and Y, X subsumes Y if:

$$P(X/Y) \geq .8, P(Y/X) < 1.$$

X subsumes Y if the documents which Y occurs in are almost (.8) a subset of the documents that X occurs in. The factor of .8 was heuristically used because an absolute condition was eliminating too many useful relationships. X is thus a parent of Y.

The set of documents to be clustered was determined by a query and the query terms were used as the initial set of terms for the monolithic cluster. This set was expanded by adding more terms via query expansion using pseudo-relevance feedback (Blind feedback, Local Context Analysis). They then used the terms and the formula above to create the hierarchies.

9.4 SUMMARY

Thesauri, semantic nets and item clusters are essential tools in Information Retrieval Systems, assisting the user in locating relevant items. They provide more benefit to the recall process than in improving precision. Thesauri, either humanly generated or statistical, and semantic nets are used to expand search statements, providing a mapping between the users vocabulary and that of the authors. The number of false hits on non-relevant items retrieved is determined by how tightly coupled the terms are in the classes. When automatic techniques are used to create a statistical thesaurus, techniques such as cliques produce classes where the

items are more likely to be related to the same concept than any of the other approaches. When a manually created thesaurus is used, human intervention is required to eliminate homonyms that produce false hits. A homonym is when a term has multiple, different meanings (e.g., the term field meaning an area of grass or an electromagnetic field). The longer (more terms) in the search statement, the less important the human intervention to eliminate homonyms. This is because items identified by the wrong interpretation of the homonym should have a low weight because the other search terms are not likely to be found in the item. When search statements are short, significant decreases in precision will occur if homonym pruning is not applied.

Item clustering also assists the user in identifying relevant items. It is used in two ways: to directly find additional items that may not have been found by the query and to serve as a basis for visualization of the Hit file. Each item cluster has a common semantic basis containing similar terms and thus similar concepts. To assist the user in understanding the major topics resulting from a search, the items retrieved can be clustered and used to create a visual (e.g., graphical) representation of the clusters and their topics. This allows a user to navigate between topics, potentially showing topics the user had not considered. The topics are not defined by the query but by the text of the items retrieved.

When items in the database have been clustered, it is possible to retrieve all of the items in a cluster, even if they were not identified by the search statement. When the user retrieves a strongly relevant item, the user can look at other items like it without issuing another search. When relevant items are used to create a new query, the retrieved hits are similar to what might be produced by a clustering algorithm. As with the term clustering, item clustering assists in mapping between a user's vocabulary and the vocabulary of the authors.

From another perspective term clustering and item clustering achieve the same objective even though they are the inverse of each other. The objective of both is to determine additional relevant items by a co-ocurrence process. A statistical thesaurus creates a cluster of terms that co-occur in the same set of items. For all of the terms within the same cluster (assuming they are tightly coupled) there will be significant overlap of the set of items they are found in. Item clustering is based upon the same terms being found in the other items in the cluster. Thus the set of items that caused a term clustering has a strong possibility of being in the same item cluster based upon the terms. For example, if a term cluster has 10 terms in it (assuming they are tightly related), then there will be a set of items where each item contains major subsets of

the terms. From the item perspective, the set of items that has the commonality of terms has a strong possibility to be placed in the same item cluster.

Hierarchical clustering of items is of theoretical interest, but has minimal practical application. The major rationale for using hierarchical clustering is to improve performance in search of clusters. The complexity of maintaining the clusters as new items are added to the system and the possibility of reduced recall are examples of why this is not used in commercial systems. Hierarchical thesauri are used in operational systems because there is additional knowledge in the human generated hierarchy. They have been historically used as a means to select index terms when indexing items. It provides a controlled vocabulary and standards between indexers.

9.5 KEYWORDS

Clustering, Thesaurus generation, Manual Clustering, Automatic Term Clustering, Complete Term Relation Method, One Pass Assignments, Item Clustering, Hierarchy of Clusters

9.6 QUESTIONS

1. Discuss the process of clustering with an emphasis on each phase of clustering.
2. What are the characteristics of classes?
3. Discuss the decisions associated with the generation of thesaurus.
4. Compare manual generated thesaurus to automated generated thesaurus.
5. Describe the process of manual clustering.
6. What is meant by automatic term clustering?
7. Explain complete term relation method.
8. Describe the process of clustering with existing clusters.
9. Explain the concept of item clustering with an example.
10. Describe the process of hierarchy of clustering with an example.

9.7 REFERENCES FROR FURTHER READINGS

Salton G, Buckley C. Text weighting approaches in automatic text retrieval Inform Proc Manag 1988;24:513-523.

Cheng D, Kannan R, Vempala S, Wang G. A divide-and merge methodology for clustering ACM Trans Data Syst 2006;31:1499-1525.

Zhao Y, Karypis G. Evaluation of hierarchical clustering algorithms for document datasets 2002. Paper presented at: CIKM'02. McLean, VA.

Steinback M, Karypis G, Kumar V. A comparison of document clustering techniques KDD Workshop on Text Mining. 2000.

Salton, G., and M. Mcgill 1983. *An Introduction to Modern Information Retrieval*. New York: McGraw-Hill.

UNIT 10: TEXT SEARCH ALGORITHMS

Structure

10.1 Introduction to Text Search Techniques

Text retrieval is a branch of information retrieval where the information is stored primarily in the form of text. Text retrieval is a critical area of study today, since it is the fundamental basis of all internet search engines. Document retrieval is sometimes referred to as, or as a branch of, Text Retrieval. Document retrieval is defined as the matching of some stated user query against a set of free-text records. These records could be any type of mainly unstructured text, such as newspaper articles, real estate records or paragraphs in a manual. User queries can range from multi-sentence full descriptions of an information need to a few words. Three classical text retrieval techniques have been defined for organizing items in a textual database, for rapidly identifying the relevant items and for eliminating items that do not satisfy the search. The techniques are full text scanning (streaming), word inversion and multi-attribute retrieval. In addition to using the indexes as a mechanism for searching text in information systems, streaming of text was frequently found in the systems as an additional search mechanism. In addition to completing a query, it is frequently used to highlight the search terms in the retrieved item prior to display.

The full text search refers to techniques for searching a single computer-stored document or a collection in a full text database. Full text search is distinguished from searches based on metadata or on parts of the original texts represented in databases. In a full text search, the search engine examines all of the words in every stored document as it tries to match search criteria. Full text searching techniques became common in online bibliographic

databases in the 1990s. Many web sites and application programs (such as word processing software) provide full-text search capabilities. Some web search engines such as AltaVista employ full text search techniques while others index only a portion of the web pages examined by its indexing system. Inversions can be used to form questions and conditional sentences, but it can also be used for emphasis. The multi-attribute retrieval explicitly models the correlations that are present between the attributes.

Basic Concept of Text streaming search system

The text scanning system allows one or more users to enter queries, and the text to be searched is accessed and compared to the query terms. The query is completed, when all the text has been accessed. One advantage of this type of architecture is that as soon as an item is identified as satisfying a query, the results can be presented to the user for retrieval. The text streaming search system is highlighted with the block diagram as follows:

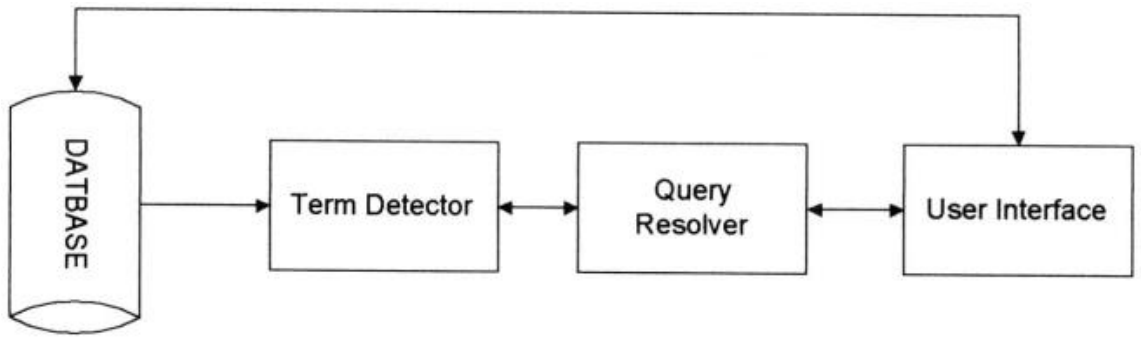


Figure 10.1: Text streaming search system

The database contains the full text of the items. The term detector is the special hardware/software that contains all of the terms being searched for and in some systems the logic between the items. It will input the text and detect the existence of the search terms. It will output to the query resolver the detected terms to allow for final logical processing of a query against an item.

The query resolver performs two functions:

i) It will accept search statements from the users, extract the logic and search terms and pass the search terms to the detector.

ii) It also accepts results from the detector and determines which queries are satisfied by the item and possibly the weight associated with hit.

The Query Resolver will pass information to the user interface that will be continually updating search status to the user and on request retrieve any items that satisfy the user

search statement. The process is focused on finding at least one or all occurrences of a pattern of text (query term) in a text stream.

Analysis: The worst case search for a pattern of m characters in a string of n characters is at least n - m + 1 or a magnitude of O(n) . Some of the original brute force methods could require O(n*m) symbol comparisons. More recent improvements have reduced the time to O(n + m).

In software systems, multiple detectors may execute at the same time. But, in the case of hardware search machines, multiple parallel search machines (term detectors) may work against the same data stream allowing for more queries or against different data streams reducing the time to access the complete database.

There are two approaches to the data stream.

i) In the first approach the complete database is being sent to the detector(s) functioning as a search of the database.

ii) In the second approach random retrieved items are being passed to the detectors.

In this second case, the idea is to perform an index search of the database and let the text streamer perform additional search logic that is not satisfied by the index search.

Examples of limits of index searches are:

- Search for stop words
- Search for exact matches when stemming is performed
- Search for terms that contain both leading and trailing “don’t cares”
- Search for symbols that are on the interword symbol list (e.g., “ , ;)

Inversions/indexes gain their speed by minimizing the amount of data to be retrieved and provide the best ratio between the total number of items delivered to the user versus the total number of items retrieved in response to a query. The inversion systems require storage overheads of 50% to 300%, of the original databases whereas the full text search function does not require any additional storage overhead. There is also the advantage where hits may be returned to the user as soon as it is found. Typically in an index system, the complete query must be processed before any hits are determined or available. Streaming systems also provide a very accurate estimate of current search status and time to complete the query.

Inversions/indexes also encounter problems in fuzzy and imbedded string query terms. It is difficult to locate all the possible index values short of searching the complete dictionary of possible terms.

The use of special hardware text search units insures a scalable environment where performance bottlenecks can be overcome by adding additional search units to work in parallel of the data being streamed. A finite state automata is used as a basis for their algorithms by most of the hardware and software text searchers.

Finite state automata

A finite state automata is a logical machine that is composed of five elements:

I - a set of input symbols from the alphabet supported by the automata

S - a set of possible states

P - a set of productions that define the next state based upon the current state and input symbol

S_0 - a special state called the initial state

S_F - a set of one or more final states from the set S

A finite state automata is represented by a directed graph consisting of a series of nodes (states) and edges between nodes represented as transitions defined by the set of productions. The symbol(s) associated with each edge defines the inputs that allow a transition from one node to another node. Figure 10.2a shows a finite state automata that will identify the character string CPU in any input stream. The automata is defined by the the automata definition in Figure 10.2b

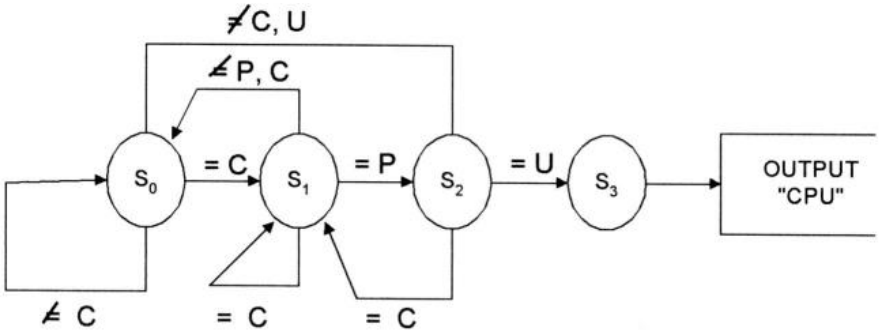


Figure 10.2a Finite State Automata

$\mathbf{I}$ = set of all alphabetic characters
$\mathbf{S}$ = set $\{S_0, S_1 S_2 S_3)$
$\mathbf{P}$ = set $\{S_0 \rightarrow S_1$ if $\mathbf{I} = C$
$S_0 \rightarrow S_0$ if $\mathbf{I} \neq C$
$S_1 \rightarrow S_2$ if $\mathbf{I} = P$
$S_1 \rightarrow S_0$ if $\mathbf{I} \neq \{P, C\}$
$S_1 \rightarrow S_1$ if $\mathbf{I} = C$
$S_2 \rightarrow S_3$ if $\mathbf{I} = U$
$S_2 \rightarrow S_1$ if $\mathbf{I} = C$
$S_2 \rightarrow S_0$ if $\mathbf{I} \neq \{C, U\}$ $\}$
$S_0 = \{ S_0 \}$
$S_F = \{ S_3 \}$

Figure 10.2b Automata Definition

The automata remains in the initial state until it has an input symbol of "C" which moves it to state S_1. It will remain in that state as long as it receives "C"s as input. If it receives a "P" it will move to S_1. If it receives anything else it falls back to the initial state. Once in state it will either go to the final state if "U" is the next symbol, go to S_1 if a "C" is received or go back to the initial state S_0, if anything else is received. The productions can also be represented by a table with the states as the rows and the input symbols that cause state transitions as each column. The states are representing the current state and the values in the table are the next state given the particular input symbol.

10.2 SOFTWARE TEXT SEARCH ALGORITHMS

In software streaming techniques, the item to be searched is read into memory and then the algorithm is applied. There are five major algorithms associated with software text search:

i) Brute force approach
ii) Knuth-Morris-Pratt
iii) Boyer-Moore
iv) Shift-OR algorithm
v) Rabin-Karp

When compared with all the algorithms, Boyer-Moore has been the fastest requiring at most O(n + m) comparisons, Knuth-Pratt-Morris and Boyer-Moore both require O(n) preprocessing of search strings.

- **Brute force approach**

It is the simplest string matching algorithm. We simply try to match the first character of the pattern with the first character of the text, and if we succeed, try to match the second character, and so on; if we hit a failure point, slide the pattern over one character and try again. When we find a match, return its starting location. The idea is to try and match the search string against the input text. If as soon as a mismatch is detected in the comparison process, shift the input text one position and start the comparison process over.

The expected number of comparisons when searching an input text string of n characters for a pattern of m characters is:

$$N_c = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right)(n - m + 1) + O(1)$$

where Nc is the expected number of comparisons

c is the size of the alphabet for the text.

Analysis: The running time of this algorithm is in O(nm).

- **Knuth-Pratt-Morris algorithm**

The Knuth–Morris–Pratt string searching algorithm (or KMP algorithm) searches for occurrences of a "word" W within a main "text string" S by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters. The algorithm was conceived in 1974 by Donald Knuth and Vaughan Pratt, and independently by James H. Morris. The three published it jointly in 1977.

The basic idea in KPM algorithm is that whenever a mismatch is detected, the previous matched characters define the number of characters that can be skipped in the input stream prior to starting the comparison process again.

For example given:

Position		1 2 3 4 5 6 7 8
Input Stream	=	a b d a d e f g
Search Pattern	=	a b d f

The first three positions of the pattern matched (a b d), then shifting one position can not find an "a" because it has already been identified as a "b". When the mismatch occurs in position 4 with a "f" in the pattern, the algorithm allows the comparison to jump at least the three positions associated with the recognized "a b d". Since the mismatch on the position could be the beginning of the search string, four positions can not be skipped. To know the number of positions to jump based upon a mismatch in the search pattern, the search pattern is pre-processed to define a number of characters to be jumped for each position. The Shift Table that specifies the number of places to jump given a mismatch is shown in Figure 10.3

Search Pattern = abcabcacab

Position in pattern	pattern character	length previous repeating substring	number of input characters to jump
1	a	0	1
2	b	0	1
3	c	0	2
4	a	0	4
5	b	1	5
6	c	2	6
7	a	3	3
8	c	4	3
9	a	0	8
10	b	1	8

Figure 10.3 Shift Characters Table

```
P  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
S  a  b  c  a  b  c  a  c  a  b
I  b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
   ↑
```

mismatch in position 1 shift one position

```
P  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
S     a  b  c  a  b  c  a  c  a  b
I  b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
            ↑
```

mismatch in position 5, no repeat pattern, skip 3 places

```
P  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
S           a  b  c  a  b  c  a  c  a  b
I  b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
            ↑
```

mismatch in position 5, shift one position

```
P  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
S              a  b  c  a  b  c  a  c  a  b
I  b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
                                     ↑
```

mismatch in position 13, longest repeating pattern is "a b c a" thus skip 3

```
P  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
S                          a  b  c  a  b  c  a  b
I  b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
```

alignment after last shift

Figure 10.4 Example of Knuth-Morris-Pratt Algorithm

Advantages:

- The running time of the KMP algorithm is optimal (O(m + n)), which is very fast.
- The algorithm never needs to move backwards in the input text T. It makes the algorithm good for processing very large files.

Disadvantages:

- Doesn't work so well as the size of the alphabets increases by which more chances of mismatch occurs.

Analysis

The running time of Knuth-Morris-Pratt algorithm is proportional to the time needed to read the characters in text and pattern. In other words, the worst-case running time of the algorithm is O(m + n) and it requires O(m) extra space. It is important to note that these quantities are independent of the size of the underlying alphabet.

❖ **Boyer-Moore string matching algorithm**

The Boyer-Moore or BM algorithm positions the pattern over the leftmost characters in the text and attempts to match it from right to left. If no mismatch occurs, then the pattern has been found. Otherwise, the algorithm computes a shift; that is, an amount by which the pattern is moved to the right before a new matching attempt is undertaken. The shift can be computed using two heuristics: the match heuristic and the occurrence heuristic.

The match heuristic is obtained by noting that when the pattern is moved to the right, it has to:

1. match the characters previously matched, and
2. bring a different character to the position in the text that caused the mismatch.

The basic idea is to compare the pattern with the text from right to left. If the text symbol that is compared with the rightmost pattern symbol does not occur in the pattern at all, then the pattern can be shifted by m positions behind this text symbol. The following example illustrates this situation.

Example:

0	1	2	3	4	5	6	7	8	9	...
a	b	b	a	d	a	b	a	c	b	a

b a b a c

c
b a b a

The first comparison d-c at position 4 produces a mismatch. The text symbol d does not occur in the pattern. Therefore, the pattern cannot match at any of the positions 0, ..., 4, since all corresponding windows contain a d. The pattern can be shifted to position 5.

- **Bad character heuristics**: It can also be applied if the bad character, i.e. the text symbol that causes a mismatch, occurs somewhere else in the pattern. Then the pattern can be shifted so that it is aligned to this text symbol. The next example illustrates this situation.

  Example:

0	1	2	3	4	5	6	7	8	9	...
a	B	b	a	b	a	b	a	c	b	a
b	A	b	a	c						
		b	a	b	a	c				

Comparison b-c causes a mismatch. Text symbol b occurs in the pattern at positions 0 and 2. The pattern can be shifted so that the rightmost b in the pattern is aligned to text symbol b.

- **Good suffix heuristics**

Sometimes the bad character heuristics fails. In the following situation the comparison ‘a-b’ causes a mismatch. An alignment of the rightmost occurrence of the pattern symbol a with the text symbol ‘a’ would produce a negative shift. Instead, a shift by 1 would be possible. However, in this case it is better to derive the maximum possible shift distance from the structure of the pattern. This method is called good suffix heuristics.

Example:

0	1	2	3	4	5	6	7	8	9	...
a	B	a	a	b	a	b	a	c	b	a
c	A	b	a	b						
		c	a	b	a	b				

The suffix ab has matched. The pattern can be shifted until the next occurence of ab in the pattern is aligned to the text symbols ab, i.e. to position 2.

In the following situation the suffix ab has matched. There is no other occurence of ab in the pattern. Therefore, the pattern can be shifted behind ab, i.e. to position 5.

Example:

0	1	2	3	4	5	6	7	8	9	...
a	b	c	a	b	a	b	a	c	b	a
c	b	a	a	b						
					c	b	a	a	b	

In the following situation the suffix bab has matched. There is no other occurence of bab in the pattern. But in this case the pattern cannot be shifted to position 5 as before, but only to position 3, since a prefix of the pattern (ab) matches the end of bab. We refer to this situation as case 2 of the good suffix heuristics.

Example:

0	1	2	3	4	5	6	7	8	9	...
a	a	b	a	b	a	b	a	c	b	a
a	b	b	a	b						
			a	b	b	a	b			

The pattern is shifted by the longest of the two distances that are given by the bad character and the good suffix heuristics.

The searching algorithm compares the symbols of the pattern from right to left with the text. The Boyer-Moore algorithm uses two different heuristics for determining the maximum possible shift distance in case of a mismatch: the "bad character" and the "good suffix" heuristics. Both heuristics can lead to a shift distance of m. For the bad character heuristics this is the case, if the first comparison causes a mismatch and the corresponding text symbol does not occur in the pattern at all. For the good suffix heuristics this is the case, if only the first comparison was a match, but that symbol does not occur elsewhere in the pattern. The

major drawback of the Boyer-Moore class of algorithms is the significant preprocessing time to set up the tables.

Analysis

The best case for the Boyer-Moore algorithm is attained if at each attempt the first compared text symbol does not occur in the pattern. Then the algorithm requires only O(n/m) comparisons.

3 If there are only a constant number of matches of the pattern in the text, the Boyer-Moore searching algorithm performs O(n) comparisons in the worst case. The proof of this is rather difficult.

4 In general $\Theta(n \cdot m)$ comparisons are necessary, e.g. if the pattern is a^m and the text a^n. By a slight modification of the algorithm the number of comparisons can be bounded to O(n) even in the general case.

5 If the alphabet is large compared to the length of the pattern, the algorithm performs O(n/m) comparisons on the average. This is because often a shift by m is possible due to the bad character heuristics.

❖ **Aho–Corasick string matching algorithm**

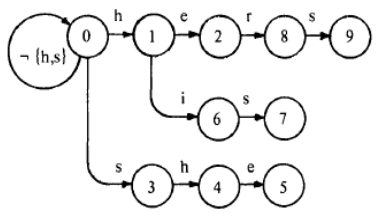
The Aho–Corasick string matching algorithm is a string searching algorithm invented by Alfred V. Aho and Margaret J. Corasick. It is a kind of dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. It matches all patterns simultaneously. The complexity of the algorithm is linear in the length of the patterns plus the length of the searched text plus the number of output matches. Since all matches are found, there can be a quadratic number of matches if every substring matches (e.g. dictionary = a, aa, aaa, aaaa and input string is aaaa).

The basic idea of the algorithm is:

- To locate all occurrences of any of finite number of keywords in a string of text.
- To construct a finite state pattern matching machine from the keywords and then use the pattern matching machine to process the text string in a single pass.

Let K= {y1,y2,…..yn} be a finite set of strings which we shall call keywords and let x be an arbitrary string which we shall call the text string. The behavior of the pattern matching machine is dictated by three functions: a goto function g , a failure function f , and an output function output.

Fig. 1. Pattern matching machine.



(a) Goto function.

i	1	2	3	4	5	6	7	8	9
$f(i)$	0	0	0	1	2	0	3	0	3

(b) Failure function.

i	$output(i)$
2	{he}
5	{she, he}
7	{his}
9	{hers}

(c) Output function.

- Goto function g : maps a pair consisting of a state and an input symbol into a state or the message fail.
- Failure function f : maps a state into a state, and is consulted whenever the goto function reports fail.
- Output function : associating a set of keyword (possibly empty) with every state.

Algorithm:

1. Start state is state 0.
2. Let s be the current state and a the current symbol of the input string x.
3. Operating cycle
    a. If g(s,a)= s' , makes a goto transition, and enters state s' and the next symbol of x becomes the current input symbol.
    b. If g(s,a)=fail, make a failure transition f. If f(s)= s', the machine repeats the cycle with s' as the current state and a as the current input symbol.

Example: Text: u s h e r s

State: 0 0 3 4 5 8 9

2

In state 4, sinceg(4,e)=5, and the machine enters state 5, and finds keywords "she" and "he" at the end of position four in text string, emits

In state 5 on input symbol r, the machine makes two state transitions in its operating cycle. Since g(5,r)=fails, M enters state 2=f(5). Then since g(2,r)=8, M enters state 8 and advances to the next input symbol. No output is generated in this operating cycle.

❖ **The Shift-Or Algorithm**

The concept used here is to represent the state of the search as a number, and each search step costs a small number of arithmetic and logical operations, provided that the numbers are large enough to represent all possible states of the search. Hence, for small patterns, we have an O(n) time algorithm using O([EI) extra space and O(m+lE[) preprocessing time, where D denotes the alphabet.

The main properties of the shift-or algorithm are:

• Simplicity: the preprocessing and the search are very simple and only bitwise logical operations, shifts and additions are used.

• Real time: the time delay to process one text character is bounded by a constant.

• No buffering: the text does not need to be stored.

The main idea is to represent the state of the search as a number.

- State=S1. 20＋S2. 21+…+Sm. 2m-1
- Tx= δ (pat1=x) . 20＋ δ (pat2=x) +…..+ δ (patm=x) . 2m-1

For every symbol x of the alphabet, where δ (C) is 0 if the condition C is true, and 1 otherwise.

Example: {a,b,c,d} be the alphabet, and ababc the pattern.

T[a]=11010,T[b]=10101,T[c]=01111,T[d]=11111

The initial state is 11111

Pattern: ababc

Text: a b d a b a b c

T[x]: 11010 10101 11111 11010 10101 11010 10101 01111

State: 11110 11101 11111 11110 11101 11010 10101 01111

For example, the state 10101 means that in the current position we have two partial matches to the left, of lengths two and four, respectively. The match at the end of the text is indicated by the value 0 in the leftmost bit of the state of the search.

❖ Rabin-Karp algorithm

A different approach to string searching is to use hashing techniques. The basic idea is to compute the signature function of each possible m-character substring in the text and check if it is equal to the signature function of the pattern.

- Suppose we are searching for 4-letter words. Then the whole (English) word fits in one (computer) word w of 4 bytes. If the current 4 bytes of the document are also in one word d, a single comparison can match the two in one step. To move along the document, shift d and add in the next character.
- For longer words, use hashing. The characters of the word and the document are combined into single hash numbers wh and dh. The hash number dh can be updated by doing a suitable sum and adding in the code for the next character.

The method of Karp and Rabin is to compare the pattern with each text window. But instead of comparing the pattern at each position with the text window, only one comparison is performed. The signature of the pattern is compared with the signature of the text window. A signature is a most unique feature.

Example: The alphabet is A = {0, ..., 9}, the pattern p = 1 3 0 8 and the signature function is the sum. The sum of 1 3 0 8 is 1 + 3 + 0 + 8 = 12.

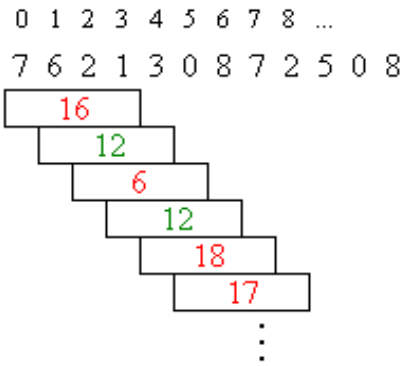


Figure 10.3 : Cross-sums in various text windows

The sum of the beginning at position 0 7 6 2 1 text window is 16 Here, therefore, do not match the pattern. In the text window in the positions 1 and 3 is the same, the cross-sum. Here it must be examined whether the pattern matches actually.

The two requirements for signature feature are:

- Collisions should be excluded as possible
- Signature must be possible to compute in constant time

A collision arises when the signature matches, the pattern is not. As seen in the above example, the checksum is not particularly well suited as a signature because it is very vulnerable to collisions.

However, enables each signature from the previous calculate in constant time: the number of people leaving the text window is subtracted, the new window for adventitious number is added. If the window position is shifted from 0 to 1, 7 to leave the window, 3 is added. The new cross There is thus out of the old as follows:

$$16\text{-}7 + 3 = 12$$

Karp and Rabin found an easy way to compute these signature functions efficiently for the signature function h(k) = k mod q, where q is a large prime. Their method is based on computing the signature function for position i given the value for position i - 1. The algorithm requires time proportional to n + rn in almost all the cases, without using extra space. This algorithm finds positions in the text that have the same signature value as the pattern, so, to ensure that there is a match, we must make a direct comparison of the substring with the pattern. This algorithm is probabilistic, but using a large value for q makes collisions 11nlikely (the probability of a random collision is O(1/q)).

Analysis

The pre-processing algorithm requires $\Theta(m)$ steps. The search algorithm requires in the worst case $\Theta(n \cdot m)$ steps, for example, if $p = a^m$ and $t = a^n$. On average, however, the algorithm has a complexity of $\Theta(n)$.

10.3 HARDWARE TEXT SEARCH SYSTEMS

Software text search is applicable to many circumstances but has encountered restrictions on the ability to handle many search terms simultaneously against the same text and limits due to I/O speeds. One approach was to have a specialized hardware machine to perform the searches and pass the results to the main computer which supported the user interface and retrieval of hits. Since the searcher is hardware based, scalability is achieved by increasing the number of hardware search devices. Another major advantage of using a hardware text search unit is in the elimination of the index that represents the document database. Other

advantages are that new items can be searched as soon as received by the system rather than waiting for the index to be created and the search speed is deterministic.

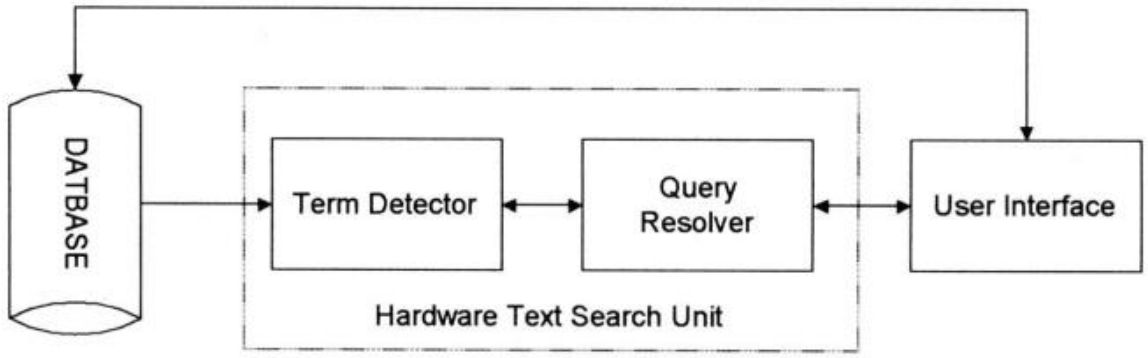


Figure 10.4 : Hardware Text Search Unit

The algorithmic part of the system is focused on the term detector. There are three approaches for implementing term detectors: parallel comparators or associative memory, a cellular structure, and a universal finite state automata .

When the term comparator is implemented with parallel comparators, each term in the query is assigned to an individual comparison element and input data are serially streamed into the detector. When a match occurs, the term comparator informs the external query resolver (usually in the main computer) by setting status flags. In some systems, some of the Boolean logic between terms is resolved in the term detector hardware (e.g., in the GESCAN machine). Instead of using specially designed comparators, specialized hardware that interfaces with computers are used to search secondary storage devices. The speed of search is based on the speed of the I/O.

One of the earliest hardware text string search units was the Rapid Search Machine developed by General Electric. The machine consisted of a special purpose search unit where a single query was passed against a magnetic tape containing the documents. A more sophisticated search unit was developed by Operating Systems Inc. called the Associative File Processor (AFP). It is capable of searching against multiple queries at the same time. Next the OSI, using a different approach, developed the High Speed Text Search (HSTS) machine. It uses an algorithm similar to the Aho-Corasick software finite state machine algorithm except that it runs three parallel state machines. One state machine is dedicated to contiguous word phrases, another for imbedded term match and the final for exact word match.

❖ **The GESCAN system**

The GESCAN system uses a text array processor (TAP) that simultaneously matches many terms and conditions against a given text stream the TAP receives the query information from the users computer and directly access the textual data from secondary storage.

The TAP consists of a large cache memory and an array of four to 128 query processors. The text is loaded into the cache and searched by the query processors. Each query processor is independent and can be loaded at any time. A complete query is handled by each query processor. Queries support exact term matches, fixed length don't cares, variable length don't cares, terms may be restricted to specified zones, Boolean logic, and proximity. A query processor works two operations in parallel; matching query terms to input text and boolean logic resolution.

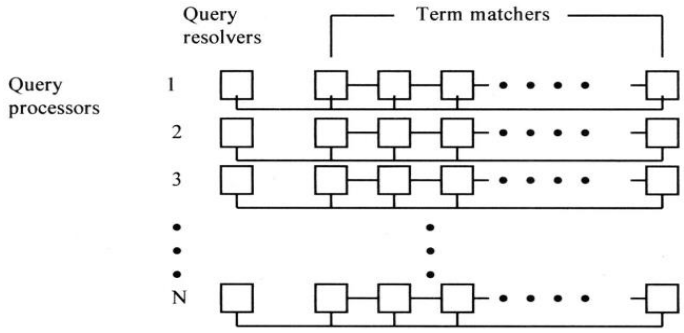


Figure 10.5: GESCAN Text Array Processor

Term matching is performed by a series of character cells each containing one character of the query. A string of character cells is implemented on the same LSI chip and the chips can be connected in series for longer strings. When a word or phrase of the query is matched, a signal is sent to the resolution sub-process on the LSI chip.

The resolution chip is responsible for resolving the Boolean logic between terms and proximity requirements. If the item satisfies the query, the information is transmitted to the users computer. The text array processor uses these chips in a matrix arrangement. Each row of the matrix is a query processor in which the first chip performs the query resolution while the remaining chips match query terms. The maximum number of characters in a query is restricted by the length of a row while the numbers of rows limit the number of simultaneous queries that can be processed.

❖ **The Fast Data Finder (FDF)**

The Fast Data Finder (FDF) is the most recent specialized hardware text search unit still in use in many organizations. It was developed to search text and has been used to search English and foreign languages. The early Fast Data Finders consisted of an array of programmable text processing cells connected in series forming a pipeline hardware search processor. The cells are implemented using a VSLI chip. In the TREC tests each chip contained 24 processor cells with a typical system containing 3600 cells. Each cell will be a comparator for a single character limiting the total number of characters in a query to the number of cells. The cells are interconnected with an 8-bit data path and approximately 20-bit control path. The text to be searched passes through each cell in a pipeline fashion until the complete database has been searched. As data is analyzed at each cell, the 20 control lines states are modified depending upon their current state and the results from the comparator. The architecture of Fast Data Finder is shown below:

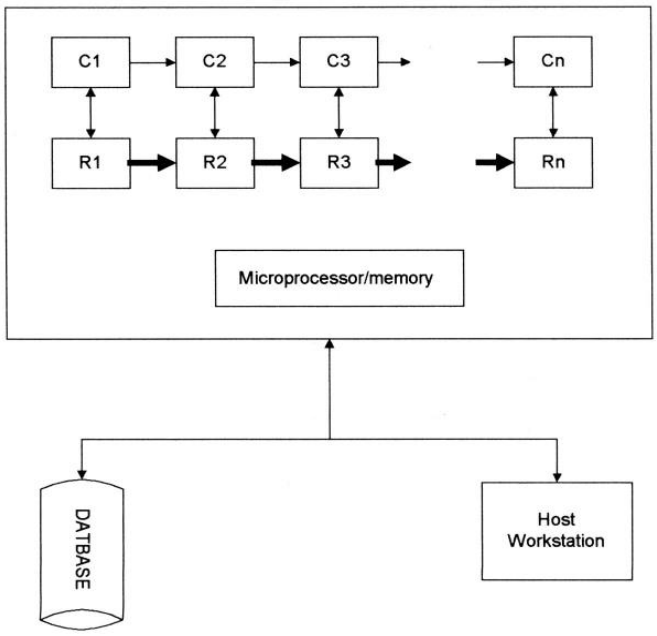


Figure 10.6 : Fast Data Finder Architecture

A cell is composed of both a register cell (Rs) and a comparator (Cs). The input from the Document database is controlled and buffered by the microprocessor/memory and feed through the comparators. The search characters are stored in the registers. The connection between the registers reflect the control lines that are also passing state information. Groups of cells are used to detect query terms, along with logic between the terms, by appropriate programming of the control lines. When a pattern match is detected, a hit is passed to the

internal microprocessor that passes it back to the host processor, allowing immediate access by the user to the Hit item. The functions supported by the Fast data Finder are: is detected, a hit is passed to the internal microprocessor that passes it back to the host processor, allowing immediate access by the user to the Hit item.

The functions supported by the Fast data Finder are:

Boolean Logic including negation

Proximity on an arbitrary pattern

Variable length "don't cares"

Term counting and thresholds

Fuzzy matching

Term weights

Numeric ranges

The Fast Data Finder is loaded with a sequence and will report back those sequences in the database whose local similarity score exceed a threshold that most closely resemble the query sequence.

Another approach for hardware searchers is to augment disc storage. The augmentation is a generalized associative search element placed between the read and write heads on the disk. The content addressable segment sequential memory (CASSM) system uses these search elements in parallel to obtain structured data from a database.

10.4 SUMMARY

Text search techniques using text scanning have played an important role in the development of Information Retrieval Systems. They currently play an important role in word processor systems (e.g., the Find function) and in Information Retrieval Systems for locating offensive terms (e.g., imbedded character strings) in the dictionary.

- Text searching algorithms can be used in various cases:
    - Small pattern: The Shift-Or Algorithm
    - Large alphabet: The Knuth-Morris-Pratt Algorithm
    - Others: The Boyer-Moore Algorithm
    - "don't care": The Shift-Or Algorithm

The Boyer-Moore algorithm is a good choice for many string-matching problems, but it does not offer asymptotic guarantees that are any stronger than those of the naive algorithm. If

asymptotic guarantees are needed, the Knuth-Morris-Pratt algorithm (KMP) is a good alternative. It is worth noting that the KMP algorithm is not a real time algorithm, and the BM algorithm needs to buffer the text. All these properties indicates that this algorithm is suitable for hardware implementation. The need for specialized hardware text search units to directly search the data on secondary storage has diminished with the growth of processing power of computers.

10.5 KEYWORDS

Hardware and software text searching, finite state automata, Brute force text search, Boyer-Moore text search, Aho-Corasick algorithm, Shift-Or Algorithm , Karp & Rabin algorithm,GE-SCAN, Fast Data Finder

10.6 QUESTIONS

1. Trade off the use of hardware versus software text search algorithms citing advantages and disadvantages of each in comparison to the other.
2. Construct finite state automata for each of the following set of terms:
    a. BIT, FIT, HIT, MIT, PIT, SIT
    b. CAN, CAR, CARPET, CASE, CASK, CAKE
    c. HE, SHE, HER, HERE, THERE, SHEAR

    Be sure to define the three sets I, S, and P along with providing the state drawing .
3. Use the Boyer-Moore text search algorithm to search for the term FANCY in
4. the text string FANCIFUL FANNY FRUIT FILLED MY FANCY.
    a. Show all of the steps and explain each of the required character shifts.
    b. How many character comparisons are required to obtain a match?
    c. Compare this to what it would take using the Knuth-Pratt-Morris algorithm
5. Use the problem defined in question three and create the GOTO, Failure and OUTPUT functions for the Aho-Corasick algorithm
6. Trace through the steps in searching for the term FANCY.
7. What are the tradeoffs between using the Aho-Corasick versus Boyer-Moore algorithms?
8. What algorithmic basis is used for the GE-SCAN and Fast Data Finder hardware text search machines? Why was this approach used over others?

10.7 REFERENCES FOR FURTHER STUDIES

R. S. Boyer and J. S. Moore, A fast string searching algorithm, Carom. ACM 20, (10), 262–272(1977).

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 32: String Matching, pp.906–932.

C. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval,. Cambridge University Press, 2008.

D. Grossman and O. Frieder. Information Retrieval: Algorithms and Heuristics

Text Information Retrieval Systems. C.T. Meadow, B.R. Boyce, D.H. Kraft, C.L. Barry. Academic Press, 2007.

W.B. Frakes and R. Baeza-Yates, Information Retrieval: Data Structures & Algorithms.. Prentice-Hall, 1992

G. Salton, Introduction to Modern Information Retrieval. M.J. McGill. McGraw-Hill, 1983.

UNIT 11: MULTIMEDIA INFORMATION RETRIEVAL

Structure

11.1 MULTIMEDIA INFORMATION RETRIEVAL

Multimedia Information retrieval is the process of satisfying a user's stated information need by identifying all relevant text, graphics, audio (speech and non speech audio), imagery or video documents or portions of documents from document collection. It is a research discipline of computer science that aims at extracting semantic information from multimedia data sources. Data sources include directly perceivable media such as audio, image and video, indirectly perceivable sources such as text, bio signals as well as not perceivable sources such as bio information, stock prices, etc.

With approximately 10 million sites on the World Wide Web, increasingly users are demanding content-based access to materials. This is evident by the advent of question answering services (e.g., www.ask.com) as well as the success of spoken language understanding and tools to support content-based access to speech. In addition, innovations are appearing in the areas of content-based access to non-speech audio, imagery and video. A separate but related body of research addresses the use of multimedia and intelligent processing to enhance the human computer interface.

The methodology of multimedia information retrieval can be organized in three groups:

- Methods for the summarization of media content (feature extraction). The result of feature extraction is a description.
- Methods for the filtering of media descriptions (for example, elimination of redundancy)
- Methods for the categorization of media descriptions into classes.

Feature Extraction Methods: Feature extraction is motivated by the complete size of multimedia objects as well as their redundancy and, possibly, noisiness. Generally, two possible goals can be achieved by feature extraction:

- **Summarization of media content.** Methods for summarization include in the audio domain, for example, Mel Frequency Cepstral Coefficients, Zero Crossings Rate, Short-Time Energy. In the visual domain, color histograms such as the MPEG-7 Scalable Color Descriptor can be used for summarization.
- **Detection of patterns by auto-correlation and/or cross-correlation**. Patterns are recurring media chunks that can either be detected by comparing chunks over the media dimensions (time, space, etc.) or comparing media chunks to templates (e.g. face templates, phrases). Typical methods include Linear Predictive Coding in the audio/biosignal domain, texture description in the visual domain and n-grams in text information retrieval

Merging and Filtering Methods: Multimedia Information Retrieval implies that multiple channels are employed for the understanding of media content. Each of these channels is described by media-specific feature transformations. The resulting descriptions have to be merged to one description per media object. Merging can be performed by simple concatenation if the descriptions are of fixed size. Variable-sized descriptions - as they frequently occur in motion description - have to be normalized to a fixed length first.

Frequently used methods for description filtering include factor analysis (e.g. by PCA), singular value decomposition (e.g. as latent semantic indexing in text retrieval) and the extraction and testing of statistical moments. Advanced concepts such as the Kalman filter are used for merging of descriptions.

Categorization Methods: Generally, all forms of machine learning can be employed for the categorization of multimedia descriptions though some methods are more frequently used in one area than another. For example, Hidden Markov models are state-of-the-art in speech recognition, while Dynamic Time Warping - a semantically related method - is state-of-the-art in gene sequence alignment. The list of applicable classifiers includes the following:

- Metric approaches (Cluster Analysis, Vector Space Model, Minkowski Distances, Dynamic Alignment)
- Nearest Neighbor methods (K-Nearest Neighbor, K-Means, Self-Organizing Map)

- Risk Minimization (Support Vector Regression, Support Vector Machine, Linear Discriminant Analysis)
- Density-based Methods (Bayes Nets, Markov Processes, Mixture Models)
- Neural Networks (Perceptron, Associative Memories, Spiking Nets)
- Heuristics (Decision Trees, Random Forests, etc.)

The selection of the best classifier for a given problem (test set with descriptions and class labels, so-called ground truth) can be performed automatically, for example, using the Weka Data Miner.

A variety of technological advances can be used to make audio less opaque, that is, provide some insight into the content of an audio file, and perhaps ways of using it as other than a monolithic block of digital data. The available methods can be roughly divided into those that assume some speech content in the audio, and those that don't.

Research challenges: The potential landscape of multimedia information retrieval is quite wide and diverse. Following are some potential areas for additional MIR research challenges.

Human Centred Methods: We should focus as much as possible on the user who may want to explore instead of search for media. It has been noted that decision makers need to explore an area to acquire valuable insight, thus experiential systems which stress the exploration aspect are strongly encouraged. Studies on the needs of the user are also highly encouraged to give us a full understanding of their patterns and desires.

Multimedia Collaboration: Discovering more effective means of human-human computer-mediated interaction is increasingly important as our world becomes more wired or wirelessly connected. In a multimodal collaboration environment, many questions remain: How do people find one another? How does an individual discover meetings/collaborations? What are the most effective multimedia interfaces in these environments for different purposes, individuals, and groups? Multimodal processing has many potential roles ranging from transcribing and summarizing meetings to correlating voices, names, and faces, to tracking individual (or group) attention and intention across media. Careful and clever instrumentation and evaluation of collaboration environments will be key to learning more about just how people collaborate.

Interactive Search and Agent Interfaces: Emergent semantics and its special case of relevance feedback methods are quite popular because they potentially allow the system to learn the goals of the user in an interactive way. Another perspective is that relevance feedback is serving as a special type of smart agent interface. Agents are present in learning environments, games, and customer service applications. They can mitigate complex tasks, bring expertise to the user, and provide more natural interaction. For example, they might be able to adapt sessions to a user, deal with dialog interruptions or follow-up questions, and help manage the focus of attention.

Neuroscience and New Learning Models*:* Observations of child learning and neuroscience suggest that exploiting information from multiple modalities (i.e., audio, imagery, haptic) reduces processing complexity. For example, researchers have begun to explore early word acquisition from natural acoustic descriptions and visual images (e.g., shape, color) of everyday objects in which mutual information appears to dramatically reduce computational complexity. This work, which exploits results from speech processing, computer vision, and machine learning, is being validated by observing mothers as play with their prelinguistic infants performing the same task.

Folksonomi: It is clear that the problem of automatically extracting content multimedia data is a difficult problem. Even in text, we could not do it completely. As a consequence, all the existing search engines are using simple keyword-based approaches or are developing approaches that have a significant manual component and address only specific areas. Another interesting finding is that, for an amorphous and large collection of information, a taxonomy-based approach could be too rigid for navigation. Since it is relatively easier to develop inverted file structures to search for keywords in large collections, people find the idea of tags attractive: by somehow assigning tags, we can organize relatively unstructured files and search.

Major Challenges: The following are the major research challenges and of particular importance to the MIR research community: (1) semantic search with emphasis on the detection of concepts in media with complex backgrounds; (2) multimodal analysis and retrieval algorithms especially to exploit the synergy between the various media, including text and context information; (3) experiential multimedia exploration systems to allow users to gain insight and explore media collections; (4) interactive search, emergent semantics, or relevance feedback systems; and (5) evaluation with emphasis on representative test sets and usage patterns.

11.2 SPOKEN LANGUAGE AUDIO RETRIEVAL

Just as a user may wish to search the archives of a large text collection, the ability to search the content of audio sources such as speeches, radio broadcasts, and conversations would be valuable for a range of applications. An assortment of techniques has been developed to support the automated recognition of speech. These have applicability for a range of application areas such as speaker verification, transcription, and command and control.

For example, Jones et al. (1997) report a comparative evaluation of speech and text retrieval in the context of the Video Mail Retrieval (VMR) project. While speech transcription word error rates may be high (as much as 50% or more depending upon the source, speaker, dictation vs. conversation, environmental factors and so on), redundancy in the source material helps offset these error rates and still support effective retrieval. In Jones et al.'s speech/text comparative experiments, using standard information retrieval evaluation techniques, speaker-dependent techniques retain approximately 95% of the performance of retrieval of text transcripts, speaker independent techniques about 75%. However, system scalability remains a significant challenge.

For example, whereas even the best speech recognition systems have on the order of 100,000 words in an electronic lexicon, text lexicons include upwards of 500,000 vocabulary words. Another challenge is the need expend significant time and effort to develop an annotated video mail corpus to support machine learning and evaluation. Some recent efforts have focused on the automated transcription of broadcast news. For example, Figure 10.1 illustrates BBN's Rough 'n' Ready prototype that aims to provide information access to spoken language from audio and video sources.

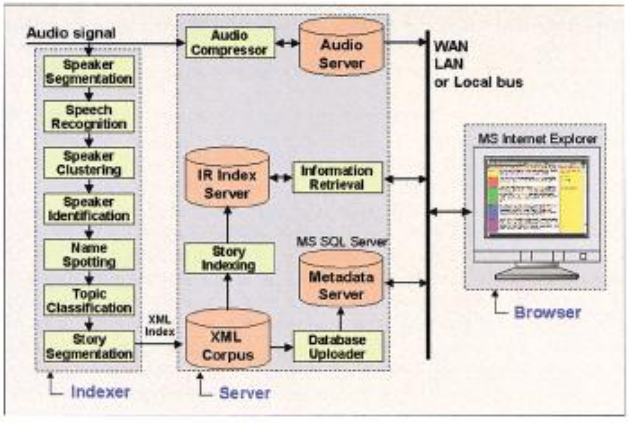


Figure 11.1: Distributed architecture of the Rough'n'Ready audio indexing and retrieval system.

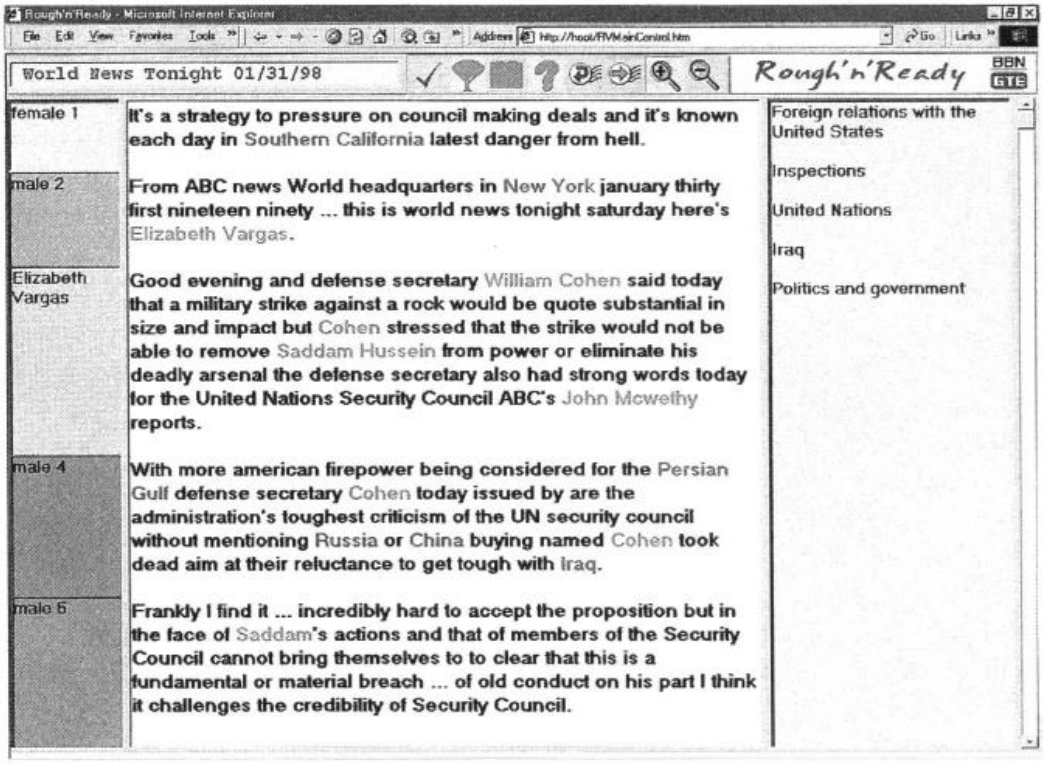


Figure 11.2. BBN's Rough and Ready

Rough'n'Ready "creates a Rough summarization of speech that is Ready for browsing." Figure 1 illustrates a January 31, 1998 sample from ABC's World News Tonight in which the left hand column indicates the speaker, the center column shows the translation with highlighted named entities (i.e., people, organizations, locations) and the rightmost column lists the topic of discussion. Rough'n'Ready's transcription is created by the BYBLOS™ large vocabulary speech recognition system, a continuous-density Hidden Markov Model (HMM) system that has been competitively tested in annual formal evaluations for the past 12 years. BYBLOS runs at 3 times real-time, uses a 60,000 word dictionary, and most recently reported word error rates of 18.8% for the broadcast news transcription task.

Additional research in broadcast news processing is addressing multilingual information access. For example, Gauvain (2000) at LIMSI reports a North American broadcast news transcription system that performs with a 13.6% word error rate and reports spoken document retrieval performance using the SDR'98 TREC-7 data. Current work is investigating broadcast transcription of German and French broadcasts. Joint research between the Tokyo Institute of Technology and NHK broadcasting is addressing transcription and topic extraction from Japanese broadcast news. The focus of Furui et al. is on improving processing by modeling filled pauses, performing on-line incremental speaker adaptation and

by using a context dependent language model that models readings of words. The language model includes Chinese characters (Kanji) and two kinds of Japanese characters (Hira-gana and Kata-kana).

Automatic Speech Recognition: Automatic Speech Recognition (ASR) is a technology rapidly coming out of the research laboratories into everyday use. While there have been decades of hard effort on the task, recent advances both in search algorithms and commonly available computing power are rapidly making ASR practical. A perfect ASR system that could quickly transcribe spoken audio documents would be an ideal solution to most audio indexing and retrieval tasks (at least for speech). Such a system would essentially reduce the audio retrieval problem to the straightforward text retrieval problem described above.

Some of the terminologies used in the spoken language audio retrieval are as follows,

Keyword Spotting: Automatically detecting words or phrases in unconstrained speech is usually termed “word spotting;" this technology is the foundation of several audio indexing efforts from a number of groups.

Sub-word indexing: Large-vocabulary ASR for audio indexing suffers, as we have seen, from several drawbacks: if a word is not present in the phonetic dictionary, it will not be recognized. Also, a language model must be used, and finding sufficient example text may not be possible. Thirdly, large-vocabulary ASR may be very expensive in terms of computation and storage (though recent advances in search algorithms are making this much less of a concern). Though this may be acceptable for typical speech recognition applications such as dictation, it is clearly unacceptable to incur several hours of computation when searching an audio corpus of similar length. To avoid some of these drawbacks, several alternatives to large-vocabulary ASR have been pursued. A common feature is the use of sub-word indexing units, such as phones or phone clusters. Typically these are smaller than words, hence there are fewer possible units, dramatically reducing the search space. An unfortunate drawback is that as units get smaller the recognition accuracy will typically decrease as well. Saving the cost of building a detailed language model will unfortunately impact recognition accuracy.

Another promising approach is “lattice-based" word spotting. A lattice is a compact representation of multiple best hypothesis generated by a phone or word recognition system. If the phone lattice is generated before need, it can then be searched extremely rapidly to find phone strings corresponding to desired query words. If the lattice contains too many

hypotheses, however, recognition accuracy will suffer from too many false alarms, as words that were not uttered can be found in a deep enough lattice.

Speaker recognition

One of the major advantages of having the actual audio signal available is the potential for recognizing the sequence of speakers. There are three consecutive components to the speaker recognition problem: speaker segmentation, speaker clustering, and speaker identification. Speaker segmentation segregates audio streams based on the speaker; speaker clustering groups together audio segments that are from the same speaker; and speaker identification recognizes those speakers of interest whose voices are known to the system. We describe each of the three components below

A. Speaker Segmentation

The goal of speaker segmentation is to locate all the boundaries between speakers in the audio signal. This is a difficult problem in broadcast news because of the presence of background music, noise, and variable channel conditions. Accurate detection of speaker boundaries provides the speech recognizer with input segments that are each from a single speaker, which enables speaker normalization and adaptation techniques to be used effectively on one speaker at a time. Furthermore, speaker change boundaries break the continuous stream of words from the recognizer into paragraph-like units that are often homogeneous in topic.

B. Speaker Clustering

The goal of speaker clustering is to identify all segments from the same speaker in a single broadcast or episode and assign them a unique label; it is a form of unsupervised speaker identification. The problem is difficult in broadcast news because of the extreme variability of the signal and because the true number of speakers can vary so widely (on the order of 1–100). We have found an acceptable solution to this problem using a bottom-up (agglomerative) clustering approach, with the total number of clusters produced being controlled by a penalty that is a function of the number of clusters hypothesized.

C. Speaker Identification

Every speaker cluster created in the speaker clustering stage is identified by gender. A Gaussian mixture model for each gender is estimated from a large sample of training data that has been partitioned by gender. The gender of a speaker segment is then determined by

computing the log likelihood ratio between the male and female models. This approach has resulted in a 2.3% error in gender detection.

In addition to gender, the system can identify a specific target speaker if given approximately one minute of speech from the speaker. Again, a Gaussian mixture model is estimated from the training data and is used to identify segments of speech from the target speaker. Any number of target models can be constructed and used simultaneously in the system to identify the speakers. To make their labeling decisions, the set of target models compete with a speaker-independent cohort model that is estimated from the speech of hundreds of speakers. Each of the target speaker models is adapted from the speaker-independent model. To ameliorate the effects of channel changes for the different speakers, cepstral mean subtraction is performed for each speaker segment whereby the mean of the feature vectors is removed before modelling

Story segmentation

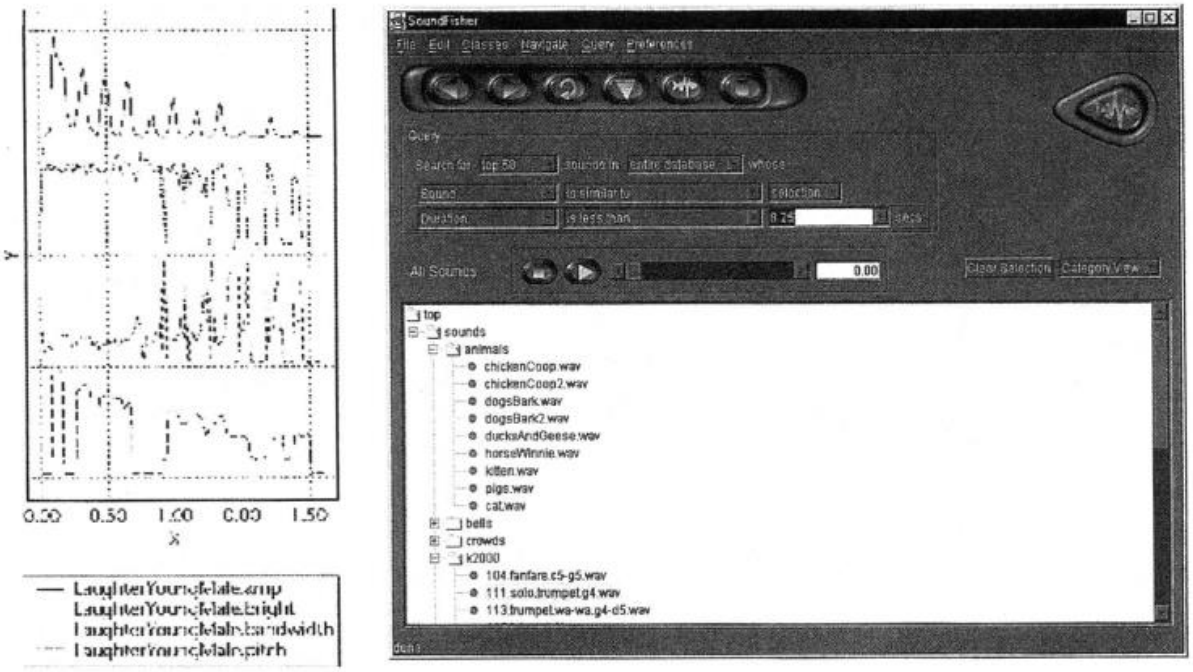
Story segmentation turns the continuous stream of spoken words into document-like units with a coherent set of topic labels assigned to each story. In Rough'n'Ready, we apply OnTopic to overlapping data windows of 200-words span, with a step size of four words between successive windows. For each data window, and for each topic of the 5500 topics known to the system, we compute the log probability of the topic given the words in the window. The list of 5500 such topic scores for each data window is pruned automatically to preserve only the top scoring (i.e., the most relevant) topics for that data window, as follows. We assume that the scores of the top scoring 100 topics are drawn from a Gaussian process, and we choose as our pruned list those topics that lie above twice the standard deviation from the mean score. The result of this process is depicted in Fig. 10, which shows the results of the topic pruning process during a transition from one story about hurricanes to another about stocks.

Fig. 11.3. The story segmentation component first chooses a few top scoring topics for each 200-word data window on a sliding basis every four words. Shown above are the chosen topics as the window passes across two stories, one about hurricanes and the other about stocks.

11.3 NON-SPEECH AUDIO RETRIEVAL.

In addition to content-based access to speech audio, noise/sound retrieval is also important in such fields as music and movie/video production. SoundFisher is a user-extensible sound classification and retrieval system, called (www.musclefish.com), that draws from several disciplines, including signal processing, psychoacoustics, speech recognition, computer music, and multimedia databases. Just as image indexing algorithms use visual feature vectors to index and match images. The authors use a vector of directly measurable acoustic features (e.g., duration, loudness, pitch, brightness) to index sounds. This enables users to search for sounds within specified feature ranges. For example, Figure 4a illustrates the analysis of male laughter on several dimensions including amplitude, brightness, bandwidth, and pitch. Figure 4b shows an enduser content-based retrieval application that enables a user to browse and/or query a sound database by acoustic (e.g., pitch, duration) and/or perceptual properties (e.g., "scratchy") and/or query by example. For example, SoundFisher supports such complex content queries as "Find all AIFF encoded files with animal or human vocal sounds that are similar to barking sounds without regard to duration or amplitude." The user can also perform a weighted query-by-value (e.g., foreground and transition with >.8 metallic and >.7 plucked aural properties and 2000 hz < average pitch < 300 hz and duration ...). The system can also be trained by example, so that perceptual properties (e.g., "scratchiness" or

"buzziness") that are more indirectly related to acoustic features can be specified and retrieved.



a. Analysis of Male Laugher. b. Content based access to audio.
Figure 11. 4. Content-based Retrieval of Non-speech Audio

Performance of the SoundFisher system was evaluated using a database of 400 widely ranging sound files (e.g., captured from nature, animals, instruments, speech). Additional requirements identified by this research include the need for sound displays, sound synthesis (a kind of query formulation/refinement tool), sound separation, and matching of trajectories of features over time.

Audio Retrieval-by-Content: Given the proliferation of audio databases on the Internet and elsewhere (some commercial sound effects libraries contain as many as 100 CDs), there is interest in doing for sound what Web search engines do for text. This requires some measure of audio similarity, which is a complicated and subjective matter. Measures of text similarity can be simple as counting the number of words in common. Most approaches to general audio retrieval take a perceptual approach, using measures derived from the audio that reflect perceptual characteristics such as brightness or loudness.

Music and MIDI retrieval: While information retrieval for text relies on simple text queries, the structure of a query for sound or music is not so obvious. Though textual descriptions can be assigned to sounds, they are not always obvious or indeed well-defined.

Researchers in the area have finessed this problem by using archives of MIDI (Musical Instrument Digital Interface) files, which are score-like representations of music intended for musical synthesizers or sequencers. Given a melodic query, then, the MIDI files can be searched for similar melodies.

When searching on non-speech audio, there are different types of queries that may be performed. For example,

- Find all sounds that are perceptually similar to a sample of laughter
- Find all songs that contain a certain melody
- Find all songs that have a piano playing

In the following section we give a brief review of related research into content-based retrieval of digital music and we follow that with some details of how content-based retrieval of music can be achieved. This section describes some of the work that has been carried out in the area of content-based retrieval of audio. The study of content-based retrieval of audio is something that is very new because the media is only now becoming widely available on the Internet. Consequently, there are very few mature pieces of research. Four systems are reported here. The first deals with the classification of different sounds, whilst the others deal with retrieving music based upon searching the melody.

Musclefish: The Musclefish system presents a system for analysing audio signals in a way that facilitates content-based retrieval. The focus in this system is at the "sound" level – acoustic and perceptual properties. Musclefish works directly with the waveform and groups sounds into different classes. This system takes the acoustical features of different sounds (such as pitch and loudness) and represents these as N-vectors. By analysing these sounds it is possible to classify audio samples, search for similar sounds, search for transitions in long pieces or convert monophonic melodies to MIDI.

Query by pitch dynamics - indexing tonal music by content: The Query by Pitch Dynamics system (QPD) provides a system for indexing a one-dimensional time series sequence of elements, based on the relative values of nearby elements rather than their absolute values. Musical notes are elements of such a time series. The problems involved with indexing this sort of time series for efficient and robust subsequence matching is addressed. The system also supports nearest neighbour queries, and clustering. In the QPD paper, the author defines a music file to be a single discrete stream of pitch values, where the duration information was ignored and multiple voices were disallowed. By using a mathematical technique, based on

the Haar wavelet transform, the closest match to a query sequence of notes from a MIDI database is returned

The New Zealand digital library melody index: The New Zealand Digital Library MELody inDEX (MELDEX) system is another "Name That Tune" system. It retrieves music based on a few notes that are sung, hummed or otherwise provided as a query to the system. The audio is transcribed into a melodic contour using pitchtracking techniques and this is used to search the database. The melodic contour uses Parsons Notation, which specifies whether the pitch of a note is above (U), below (D) or the same (S) as the previous note. Using approximate string matching the query melody is matched to the best result from the melodies in the database

Query by Humming: The Query by Humming system is another musical information retrieval system. It takes in a hummed query via a microphone and using a pitch tracker it extracts the notes from the query. This is then compared against a database of songs to return the songs that match the query.

Of the working content-based musical Information Retrieval systems examined, the Musclefish system is the only one concerned with the properties of the acoustical waveform. This system is concerned with grouping and classifying sounds. The other three systems studied, namely Query by Pitch Dynamics, the New Zealand Digital Library Melody Index and Query by Humming, are concerned with the retrieval of music by querying on the melody. This is done by evaluating the melodic contour, rather than searching directly on the notes played. All three systems use MIDI files as the database store and this suggests that MIDI, as a storage format for musical information, be worth further investigation.

Building a better system

Each of the different systems that currently exist have their strengths and their weaknesses. By combining the strengths of each of the systems, it is possible to develop a new and better system.

Query Input

An ideal system should include both a graphical and audio input. The graphical input should provide a means for inputting different pitches in a graphical fashion. The system should not be limited to the number of notes that can be input, and it should also allow the inclusion of temporal information by dragging notes around the screen. The interface should include a

mechanism for the user to hear how the sequence of notes sound before submitting the melody.

Searching on the melody

The melodies of a piece of music can be searched by using the dynamics of the melodic contour. A good system should incorporate means of accurately describing the melodic contour of a melody piece. It should be easy to search for the closest matching melody in an efficient and accurate way. Parson's Notation describes a means of defining a melodic contour. This can be searched using approximate string matching techniques.

Searching using temporal information

The system should support the temporal nature of music. The duration of each particular note is almost as important as the actual note being played. By sampling the duration of the notes played, as demonstrated in Figure 6, all notes can be made to be of equal duration. The tempo should also be used in the search. Pieces of music where the tempo is close to the query should be returned as more likely results. The system should allow the person querying the database to provide boundaries for the tempo, e.g. return songs with a certain melody where the tempo lies between 100 BPM and 130 BPM.

Searching using other attributes

It may be important to consider the velocity of notes. This could be done by using a threshold value under which any notes with a velocity less than the threshold are ignored. The system should support the specification or recognition of particular instruments. This could be a combination of a selection by the user making the query or automatic recognition in software on the query melody

Returning Results

The results should be returned in a ranked order where the most likely pieces of music are at the top of the list. The interface should be powerful enough to let the user know which attributes of the piece of music most closely match the query. The system should allow the user to hear the music returned and in particular the piece which matches the query to allow the user to narrow the search.

11.4 SUMMARY

Evaluation of Information Retrieval Systems is essential to understand the source of weaknesses in existing systems and tradeoffs between using different algorithms. The standard measures of Precision, Recall, and Fallout have been used for the last twenty-five years as the major measures of algorithmic effectiveness.

With the insertion of information retrieval technologies into the commercial market and ever growing use on the Internet, other measures will be needed for real time monitoring the operations of systems. The measures to date are optimal from a system perspective, and very useful in evaluating the effect of changes to search algorithms. What are missing are the evaluation metrics that consider the total information retrieval system, attempting to estimate the system's support for satisfying a search versus how well an algorithm performs.

11.5 KEYWORDS

Multimedia information retrieval, Rough and Ready, Automatic Speech Recognition, Story segmentation, Audio retrieval, Music retrieval

11.6 QUESTIONS

1. Define Multimedia Information retrieval.
2. Explain the methodologies for multimedia information retrieval?
3. Explain the challenges in multimedia information retrieval?
4. Write a note on BBC's Rough'n'Ready audio indexing and retrieval system.
5. Describe different steps involved in Speaker recognition.
6. Explain non-speech audio retrieval.
7. Explain different types of queries involved in non-speech audio retrieval.

11.7 REFERENCES FOR FURTHER READING/STUDIES

[1] H Eidenberger. " Fundamental Media Understanding ", atpress, 2011,

[2] A Del Bimbo. " Visual Information Retrieval ", Morgan Kaufmann, 1999.

[3] MS Lew (Ed.). " Principles of Visual Information Retrieval ", Springer, 2001.

[4] http://en.wikipedia.org/wiki/Multimedia_Information_Retrieval retrieved on 20-11-2012

UNIT 12: HYPOTHETICAL SYSTEMS

Structure

12.0 INTRODUCTION

This chapter discusses the retrieval of a range of classes of media including graphics, imagery, and video. When dealing with imagery, audio, or video, we must utilize techniques that process different elements. In audio, this might mean phonemes (or basic units of sound) and their properties (e.g., loudness, pitch), in imagery this might include principle components such as color, shape, texture, and location, and in video this might include camera position and movement in addition to imagery and audio elements.

12.1 GRAPH RETRIEVAL

An important media class is graphics that include tables and charts (e.g., column, bar, line, pie, scatter). Graphs are constructed from more primitive data elements such as points, lines, and labels. An example of a graph retrieval system is Sagebook. The SageBook enables both search and customization of stored data graphics. Sagebook supports datagraphic query, representation (i.e., content description), indexing, search, and adaptation capabilities.

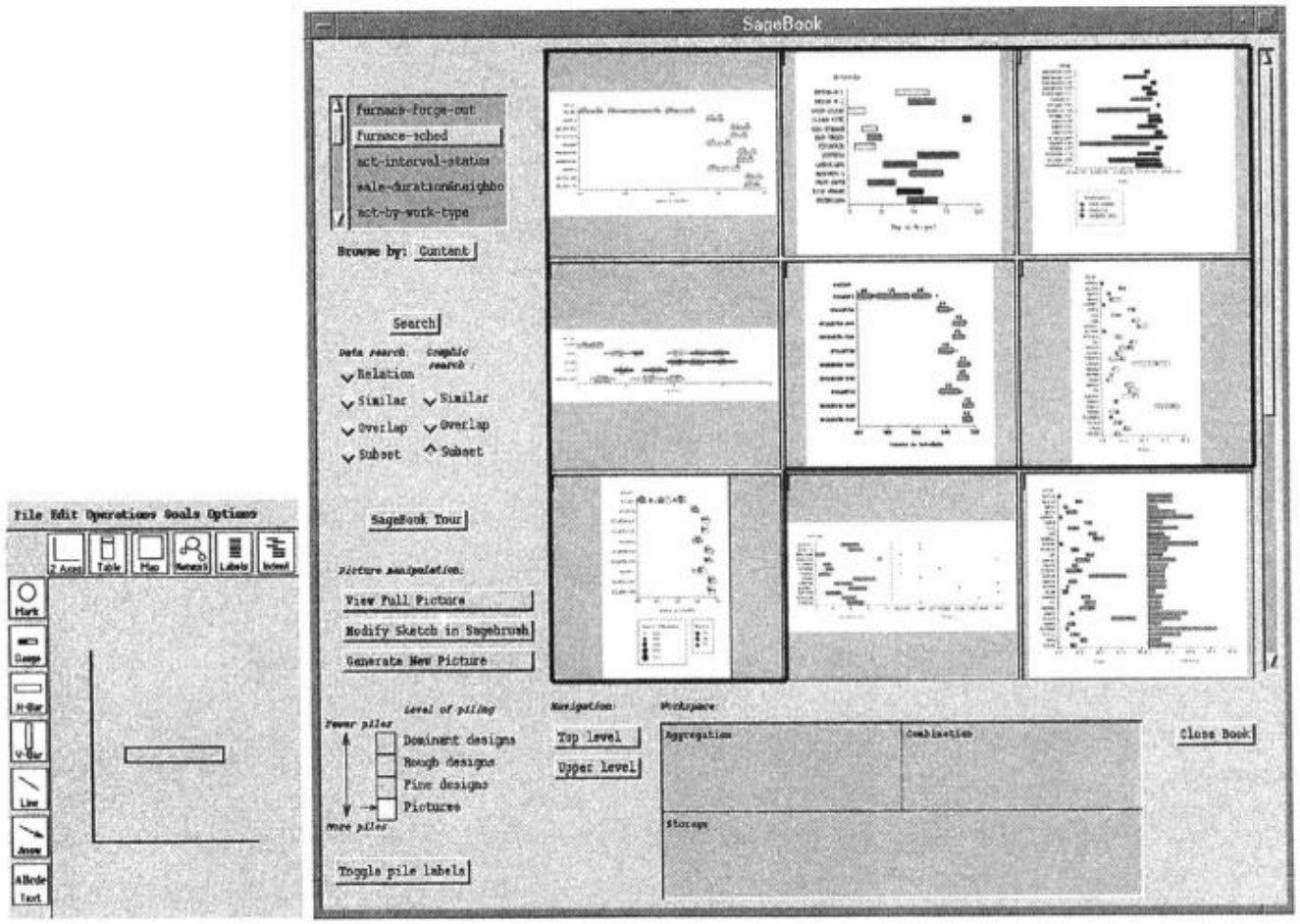


Figure 12.1. SageBrush Query Interface and SageBook display of retrieved relevant graphics

In the bottom left hand side of Figure 12.1, the queries are formulated via a graphical direct-manipulation interface (called SageBrush) by selecting and arranging spaces (e.g., charts, tables), objects contained within those spaces (e.g., marks, bars), and object properties (e.g., color, size, shape, position). The right hand side of Figure 12.1 displays the relevant graphics retrieved by matching the underlying content and/or properties of the graphical query at the bottom left of Figure 12.1 with those of graphics stored in a library. Both exact matching and similarity based matching is performed on either graphical elements (or graphemes) as well as on the underlying data represented by the graphic.

For example, in the query and responses in Figure 12.1, for two graphemes to match, they must be of the same class (i.e. bars, lines, marks) as well as use the same properties (i.e. color, shape, size, width) to encode data. The matches returned are sorted according to their degree of similarity to the query based on the match criteria.

In Figure 12.1, all the data-graphics returned by a "close graphics matching strategy" (i.e., they are all of type "chart", have exactly one space in the graphic, and contain graphemes of type *horizontal interval bar*) are highlighted in the Figure.

The retrieved data-graphics can be manually adapted. SageBook maintains an internal representation of the syntax and semantics of data-graphics, which includes spatial relationships between objects, relationships between data domains (e.g., interval, 2D coordinate), and the various graphic and data attributes. Search is performed both on graphical and data properties, with three and four alternative search strategies, respectively, to enable varying degrees of match relaxation. Several data-graphic grouping techniques based on data and graphical properties were designed to enable clustering for browsing large collections. Finally, SageBook provides automatic adaptation techniques that can modify the retrieved graphic (e.g., eliminating graphical elements) that do not match the specified query.

The graphics retrieval can also be performed based on the content. It may enable new capabilities in a broad range of domains beyond business graphics. For example, graphics play a predominant role in domains such as cartography (terrain, elevation, features), architecture (blueprints), communications and networking (routers and links), systems engineering (components and connections) and military campaign planning (e.g., forces and defenses overlaid on maps). In each of these cases graphical elements, their properties, relations, and structure, can be analyzed for retrieval purposes.

12.2 IMAGERY RETRIEVAL

Indexing and search of not only the metadata associated with the imagery (e.g., captions, annotations) was the need of the researchers but also retrieval directly on the content of the imagery.

Query By Image Content (QBIC) system exemplifies this imagery attribute indexing approach. QBIC supports access to imagery collections on the basis of visual properties such as color, shape, texture, and sketches (viewing from the Internet will show colors described in the text.). This approach provides query facilities for specifying color parameters, drawing desired shapes, or selecting textures replace the traditional keyword query found in text retrieval.

For example, Figure 12.2a illustrates a query to a database of all US stamps prior to 1995 in which QBIC is asked to retrieve red images. The "red stamps" results are displayed in Figure 12.2b. If there

are text captions associated with the imagery these of course can be exploited. For example, if this search is further refined by adding the keyword “president” we obtain the results in which all stamps are both red in color and are related to “president”.

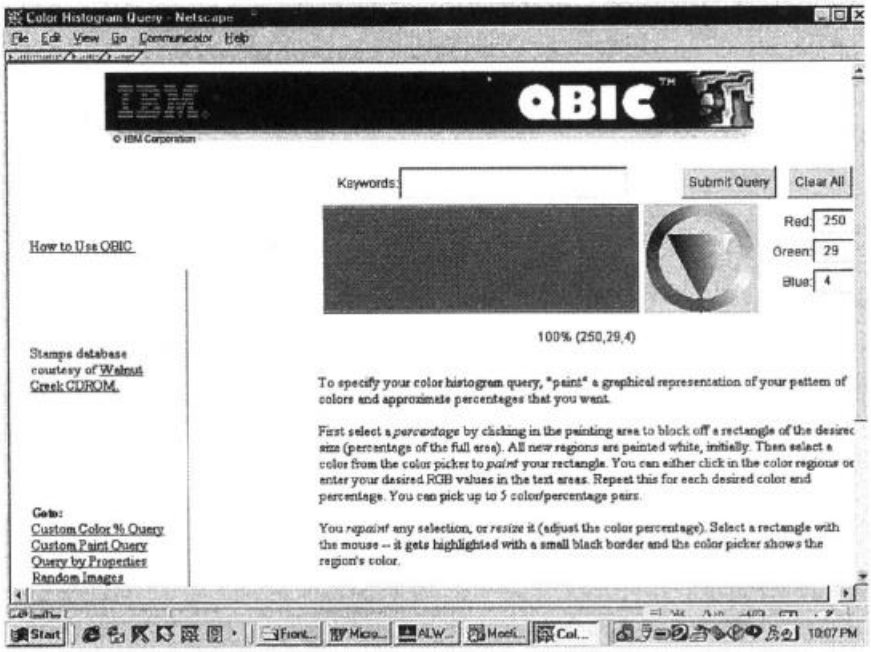


Figure12.2a. QBIC Query by Color red

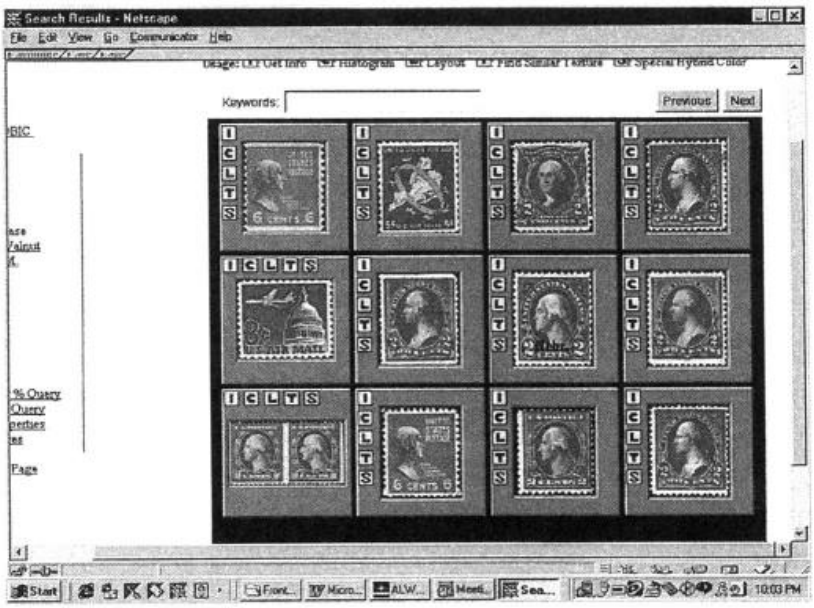


Figure 12.2b. Retrieved red stamps

Using QBIC the user can also specify queries such as “find images with a coarsely textured, red round object and a green square”.The automated and semi automated object outlining tools are

developed (e.g., foreground/background models to extract objects) to facilitate database population.

12.3 Text Retrieval

The content based imagery was applied to provide access to video retrieval. It helps in performing shot detection, extract a representative frame (r-frame, sometimes called keyframe) for each shot, and derive a layered representation of moving objects. This enables queries such as "find me all shots panning left to right" which yield a list of relevancy ranked r-frames (which acts as a thumbnail), selection of which retrieves the associated video shot.

An advanced video retrieval solution could identify the text present in the video, recognize the text, compute the similarity between the query string and pre-indexed textual information present in the video. Moreover for many of the languages, we do not have OCRs available for decoding the textual content in the frames. Since we do not have OCRs available to work effectively on the text in the video data, we use text images to index the videos.

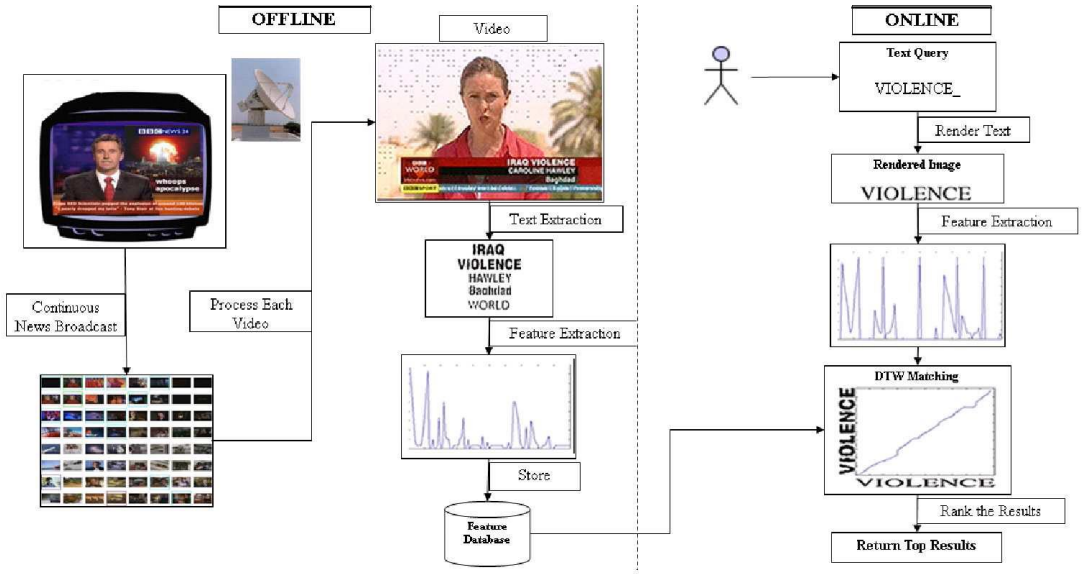


Figure 12.3: A conceptual diagram of the text-image-based video retrieval

The text blocks are extracted in video frames. Extraction of text information involves detection, localization, enhancement and recognition of the textual content in the video frames]. This method involves a frame by frame processing on the entire video for locating textual blocks. Each frame is divided into regions of size N x N, where N depends on the size

of the frame. For our experiments, we divided a 320 x 240 frame into 25 parts. These regions are separated into different classes using Multi-level Thresholding. These classes of pixels are then arranged into planes based on the pixel and location similarities. On each of these planes, we perform connected component analysis followed by XY-cut to detect and locate the text. Further, textual regions are used for indexing and retrieval of the video. Words are matched at the image-level, without any explicit textual representation for providing the access to the video database.

During the online phase, a search query is entered through a graphical user interface. This query string is rendered as an image and the corresponding set of features is extracted. These features are same as those employed in the offline process. A matching algorithm then computes the degree of similarity between the features of the search query and those present in the feature database. The results are then ranked based on the similarity measure. Word form variations are handled using a partial matching method, based on Dynamic Time Warping(DTW).

12.4 Video Retrieval

In a database of videos, one can query for relevant videos with example images, as it is popular for content based image retrieval. Several approaches have been reported for indexing and retrieval of video collections. They model spatial and temporal characteristics of the video for representation of the video content. In spatial domain, feature vectors are computed from different parts of the frames and their relationship is encoded as a descriptor. The temporal analysis partitions the video into basic elements like frames, shots, scenes or video-segments. Each of the video segments is then characterized by the appearance and dynamism of the video content. It is often assumed that features like, histograms, moments, texture and motion vectors, can describe the information content of the video clip.

Audio and the textual content in videos can be of immense use in indexing. Textual information is present as captions appearing on the video or printed/handwritten text in the scene. If we can detect and recognize the textual content, it can be effectively used for characterizing the video clips, as a complementary measure to the visual appearance based features.

Broadcast News Navigator (BNN) system is a web-based tool that automatically captures, annotates segments, summarizes and visualizes stories from broadcast news video. BNN integrates text, speech, and image processing technologies to perform multi-stream analysis of video to support content-based search and retrieval. BNN addresses the problem of time consuming, manual video acquisition/annotation techniques that frequently result in inconsistent, error-full or incomplete video catalogues.

In Figure 12.4a the user has selected to search all news video sources for a 2 week period (27 February to 12 March, 2000) using free text as well as person and location tags. BNN also supports simple browsing of stories during particular time intervals or from particular sources. A useful facility in this regard is the ability to display a graph of named entity frequency over time. In addition, the user can automatically data mine the named entities in the analyzed stories using the "search for correlations"

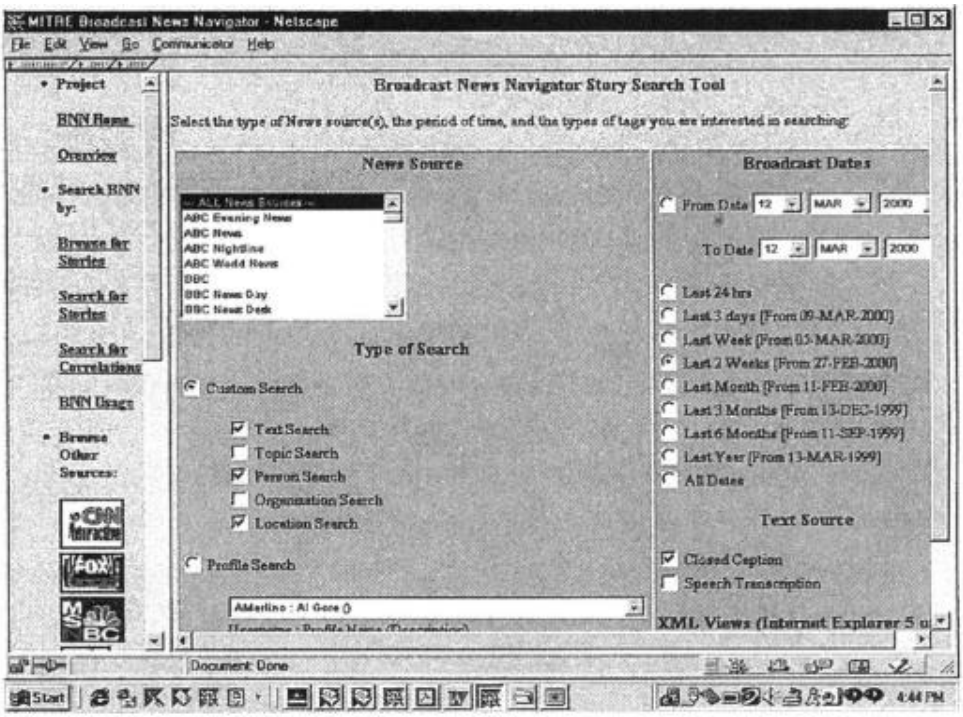


Figure 12.4a. Initial Query Page

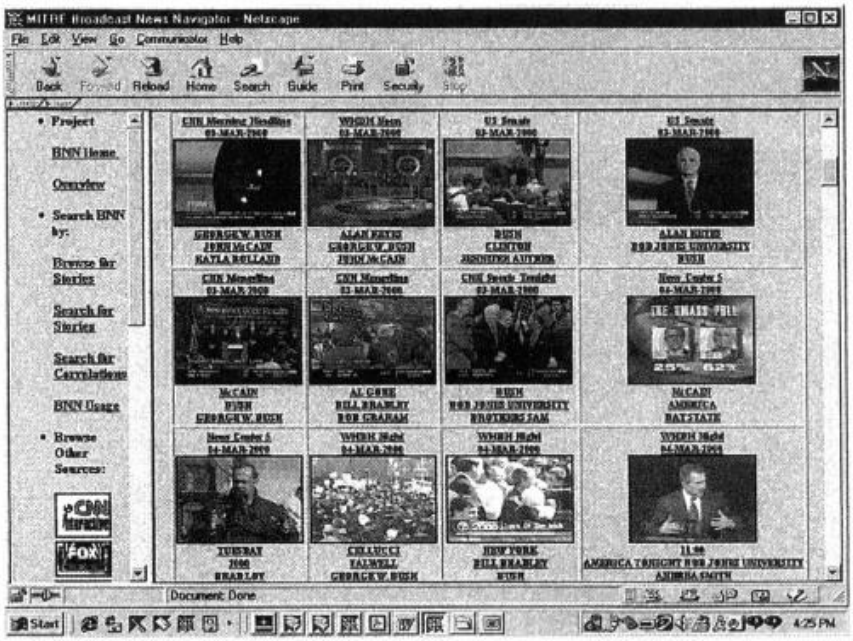


Figure 12.4b: "George Bush" Stories

BNN users could improve their retrieval performance by looking at only the 3 most frequent named entities (i.e., people, organizations, and locations) in the story (as in Figure 12.4b) rather than looking at the story details.

12.5 SUMMARY

Imagery retrieval is presently based on shallow image feature analyses (e.g., color, shape, texture) and so integration of these results with those from deeper semantic analysis of text documents remains a challenge. Higher level intentional models of media require even more sophisticated analytic methods but promise even deeper representations of media and correspondingly more powerful retrieval.

12.6 KEYWORDS

Graph Retrieval, Imagery Retrieval, Text Retrieval, Video Retrieval

12.7 QUESTIONS

1. What elements in video can be used to index the content?
2. Define face detection, face recognition, and face retrieval.
3. List three applications for content-based video?

4. What new media do you believe will appear in the future and benefit from content based retrieval?
5. What new application areas do you envision being enabled by content based multimedia retrieval?
6. Explain the various approaches of graph retrieval.
7. Explain the various approaches of imagery retrieval.
8. Explain the various approaches of video retrieval.

12.8 REFERENCES FOR SELF STUDY

Frakes, William B. (1992). Information Retrieval Data Structures & Algorithms. Prentice-Hall, Inc.. ISBN 0-13-463837-9.

G. Salton, E. Fox, and H. Wu. Extended Boolean Information Retrieval. Communications of the ACM, 1983, 26(11): 1022-1036.

Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich (2008). Introduction to Information Retrieval. Cambridge University Press.

Korfhage, Robert R. (1997). Information Storage and Retrieval. Wiley. pp. 368 pp.. ISBN 978-0-471-14338-3.

UNIT-13: USER SEARCH TECHNIQUES

Structure

13.0 INTRODUCTION

This unit focuses on how search is performed. To understand the search process, it is first necessary to look at the different binding levels of the search statement entered by the user to the database being searched. The selection and ranking of items is accomplished via similarity measures that calculate the similarity between the user's search statement and the weighted stored representation of the semantics in an item. Relevance feedback can help a user enhance search by making use of results from previous searches. This technique uses information from items judged as relevant and non-relevant to determine an expanded search statement.

13.1 SEARCH STATEMENTS AND BINDING

Search statements are the statements of an information need generated by users to specify the concepts they are trying to locate in items. The search statement uses traditional Boolean logic and/or Natural Language. In generation of the search statement, the user may have the ability to weight (assign an importance) to different concepts in the statement. At this point the binding is to the vocabulary and past experiences of the user. Binding in this sense is when a more abstract form is redefined into a more specific form. The search statement is the user's attempt to specify the conditions needed to subset logically the total item space to that cluster of items that contains the information needed by the user.

The next level of binding comes when the search statement is parsed for use by a specific search system. The search system translates the query to its own meta language. This process is similar to the indexing of item. For example, statistical systems determine the processing tokens of interest and the weights assigned to each processing token based upon frequency of

occurrence from the search statement. Natural language systems determine the syntactical and discourse semantics using algorithms similar to those used in indexing. Concept systems map the search statement to the set of concepts used to index items.

The final level of binding comes as the search is applied to a specific database. This binding is based upon the statistics of the processing tokens in the database and the semantics used in the database. This is especially true in statistical and concept indexing systems. Some of the statistics used in weighting are based upon the current contents of the database. Some examples are Document Frequency and Total Frequency for a specific term. Frequently in a concept indexing system, the concepts that are used as the basis for indexing are determined by applying a statistical algorithm against a representative sample of the database versus being generic across all databases. Natural Language indexing techniques tend to use the most corpora-independent algorithms. Figure 1.1 illustrates the three potential different levels of binding. Parentheses are used in the second binding step to indicate expansion by a thesaurus.

INPUT	Binding
"Find me information on the impact of the oil spills in Alaska on the price of oil"	User search statement using vocabulary of user
impact, oil (petroleum), spills (accidents), Alaska, price (cost, value)	Statistical system binding extracts processing tokens
impact (.308), oil (.606), petroleum (.65), spills (.12), accidents (.23), Alaska (.45), price (.16), cost (.25), value (.10)	Weights assigned to search terms based upon inverse document frequency algorithm and database

Figure 13.1 Examples of Query Binding

13.2 SIMILARITY MEASURES AND RANKING

Searching in general is concerned with calculating the similarity between a user's search statement and the items in the database. Although many of the older systems are unweighted, the newer classes of Information Retrieval Systems have logically stored weighted values for the indexes to an item. The similarity may be applied to the total item or constrained to logical passages in the item. For example, every paragraph may be defined as a passage or every 100 words.

In this case, the similarity will be to the passages versus the total item. Rather limiting the definition of a passage to a fixed length size, locality based similarity allows variable length

passages (neighbour hoods) based upon similarity of content . In results presented at TREC-4, it was discovered that passage retrieval makes a significant difference when search statements are long (hundreds of terms) but does not make a major difference for short queries. The lack of a large number of terms makes it harder to find shorter passages that contain the search terms expanded from the shorter queries.

Once items are identified as possibly relevant to the user's query, it is best to present the most likely relevant items first. This process is called "ranking." Usually the output of the use of a similarity measure in the search process is a scalar number that represents how similar an item is to the query.

13.2.1 Similarity Measures

A variety of different similarity measures can be used to calculate the similarity between the item and the search statement. A characteristic of a similarity formula is that the results of the formula increase as the items become more similar. The value is zero if the items are totally dissimilar.

Sum of the Products Similarity Measure

An example of a simple "sum of the products" similarity measure to determine the similarity between documents for clustering purposes is:

$$SIM(Item_i, Item_j) = \Sigma\ (Term_{i,k})\ (Term_{j,k})$$

This formula uses the summation of the product of the various terms of two items when treating the index as a vector. If **Itemj** is replaced with **Queryj** then the same formula generates the similarity between every Item and **Queryj**. The problem with this simple measure is in the normalization needed to account for variances in the length of items. Additional normalization is also used to have the final results come between zero and +1 (some formulas use the range -1 to +1). One of the originators of the theory behind statistical indexing and similarity functions was Robertson and Spark Jones (Robertson-76). Their model suggests that knowledge of terms in relevant items retrieved from a query should adjust the weights of those terms in the weighting process. They used the number of relevant documents versus the number of non-relevant documents in the database and the number of relevant documents having a specific query term versus the number of non-relevant

documents having that term to devise four formulas for weighting. This assumption of the availability of relevance information in the weighting process was later relaxed by Croft and Harper (Croft-79). Croft expanded this original concept, taking into account the frequency of occurrence of terms within an item producing the following similarity formula (Croft-83):

$$\mathbf{SIM(DOC_i, QUERY_j)} = \sum_{i=1}^{Q} (C + IDF_i) * f_{i,j})$$

where C is a constant used in tuning **IDFi**, is the inverse document frequency for term "i" in the collection and

$$f_{i,j} = \mathbf{K + (K - 1)\ TF_{i,j}/maxfreq_j}$$

where K is a tuning constant, **TFi,j** is the frequency of **Termi** "i" **itemj** and **maxfreqj** is the maximum frequency of any term in item "j." The best values for K seemed to range between 0.3 and 0.5.

Similarity Measure based on Cosine formula

Another early similarity formula was used by Salton in the SMART system (Salton-83). Salton treated the index and the search query as n dimensional vectors. To determine the "weight" an item has with respect to the search statement, the Cosine formula is used to calculate the distance between the vector for the item and the vector for the query:

$$\mathbf{SIM(DOC_i, QUERY_j)} = \frac{\sum_{k=1}^{n} (DOC_{i,k} * QTERM_{j,k})}{\sqrt{\sum_{k=1}^{n} (DOC_{i,k})^2 * \sum_{k=1}^{n} (QTERM_{j,k})^2}}$$

where **DOCj,k** is the kth term in the weighted vector for Item "i" and **QTERMj,k** is the kth term in query "j." The Cosine formula calculates the Cosine of the angle between the two vectors. As the Cosine approaches "1," the two vectors become coincident (i.e., the term and the query represent the same concept). If the two are totally unrelated, then they will be orthogonal and the value of the Cosine is "0." What is not taken into account is the length of the vectors. For example, if the following vectors are in a three dimensional (three term) system:

Item = (4, 8, 0)
Query 1 = (1, 2, 0)
Query 2= (3, 6, 0)

then the Cosine value is identical for both queries even though Query 2 has significantly higher weights in the terms in common. To improve the formula, Salton and Buckley (Salton-88) changed the term factors in the query to:

$$QTERM_{i,k} = (0.5 + (0.5\ TF_{i,k}/maxfreq_k)) * IDF_i$$

where $TF_{i,k}$ is the frequency of term "i" in query "k," $maxfreq_k$ is the maximum frequency of any term in query "k" and IDF_i is the inverse document frequency for term "i". In the most recent evolution of the formula, the IDF factor has been dropped (Buckley-96).

Jaccard and the Dice Similarity Measure

Two other commonly used measures are the Jaccard and the Dice similarity measures (Rijsbergen-79). Both change the normalizing factor in the denominator to account for different characteristics of the data. The denominator in the Cosine formula is invariant to the number of terms in common and produces very small numbers when the vectors are large and the number of common elements is small. In the Jaccard similarity measure, the denominator becomes dependent upon the number of terms in common. As the common elements increase, the similarity value quickly decreases, but is always in the range -1 to +1.

The Jaccard formula is :

$$SIM(DOC_i, QUERY_j) = \frac{\sum_{k=1}^{n} (DOC_{i,k} * QTERM_{j,k})}{\sum_{k=1}^{n} DOC_{i,k} + \sum_{k=1}^{n} QTERM_{j,k} - \sum_{k=1}^{n} (DOC_{i,k} * QTERM_{j,k})}$$

The Dice measure simplifies the denominator from the Jaccard measure and introduces a factor of 2 in the numerator. The normalization in the Dice formula is also invariant to the number of terms in common.

$$SIM(DOC_i, QUERY_j) = \frac{2 * \sum_{k=1}^{n} (DOC_{i,k} * QTERM_{j,k})}{\sum_{k=1}^{n} DOC_{i,k} + \sum_{k=1}^{n} QTERM_{j,k}}$$

Figure 13.2 shows how the normalizing denominator results vary with the commonality of terms. For the Dice value, the numerator factor of 2 is divided into the denominator. Notice that as long as the vector values are same, independent of their order, the Cosine and Dice normalization factors do not change. Also notice that when there are a number of terms in common between the query and the document, that the Jaccard formula can produce a negative normalization factor.

It might appear that similarity measures only apply to statistical systems where the formulas directly apply to the stored indexes. In the implementation of Natural Language systems, also

weighted values come from statistical data in conjunction with the natural language processing stored as indexes. Similarity algorithms are applied to these values in a similar fashion to statistical systems. But in addition to the similarity measures, constructs are used at the discourse level to perform additional filtering of the items.

Use of a similarity algorithm returns the complete data base as search results. Many of the items have a similarity close or equal to zero (or minimum value the similarity measure produces). For this reason, thresholds are usually associated with the search process. The threshold defines the items in the resultant Hit file from the query. Thresholds are either a value that the similarity measure must equal or exceed or a number that limits the number of items in the Hit file.

QUERY = (2, 2, 0, 0, 4)
DOC1 = (0, 2, 6, 4, 0)
DOC2 = (2, 6, 0, 0, 4)

	Cosine	Jaccard	Dice
DOC1	36.66	16	20
DOC2	36.66	–12	20

Figure 13.2 Normalizing Factors for Similarity Measures

A default is always the case where the similarity is greater than zero. Figure 1.3 illustrates the threshold process. The simple "sum of the products" similarity formula is used to calculate similarity between the query and each document. If no threshold is specified, all three documents are considered hits. If a threshold of 4 is selected, then only DOC1 is returned.

Vector:	American, geography, lake, Mexico, painter, oil, reserve, subject
DOC1	geography *of* Mexico *suggests* oil reserves *are available*
	vector (0, 1, 0, 2, 0, 3, 1, 0)
DOC2	American geography *has* lakes *available everywhere*
	vector (1, 3, 2, 0, 0, 0, 0, 0)
DOC3	painters *suggest* Mexico lakes *as* subjects
	vector (0, 0, 1, 3, 3, 0, 0, 2)

QUERY oil reserves *in* Mexico

vector (0, 0, 0, 1, 0, 1, 1, 0)

SIM(Q, DOC1) = 6, SIM (Q, DOC2) = 0, SIM(Q, DOC3) = 3

Figure 1.3 Query Threshold Process

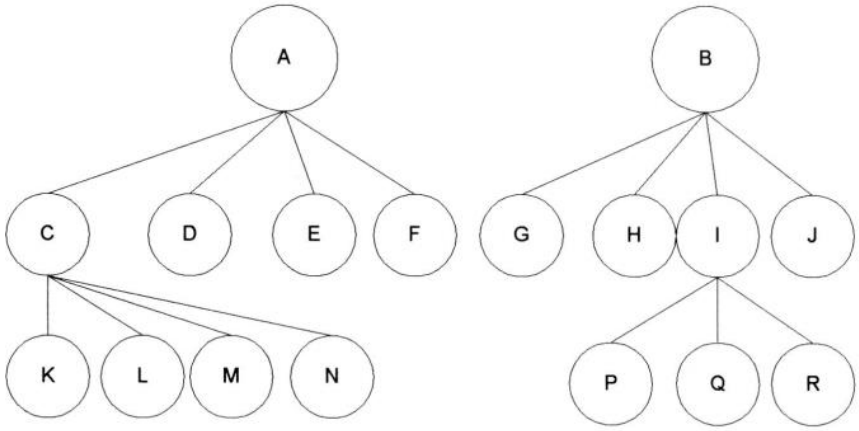


Figure 13.4 Item Cluster Hierarchies

One special area of concern arises from search of clusters of terms that are stored in a hierarchical scheme. The items are stored in clusters that are represented by the centroid for each cluster. Figure 1.4 shows a cluster representation of an item space. In Figure 1.4, each letter at the leaf (bottom nodes) represent an item (i.e., K, L, M, N, D, E, F, G, H, P, Q, R, J). The letters at the higher nodes (A, C, B, I) represent the centroid of their immediate children nodes. The hierarchy is used in search by performing a top-down process. The query is compared to the centroids "A" and "B." If the results of the similarity measure are above the threshold, the query is then applied to the nodes' children. If not, then that part of the tree is pruned and not searched. This continues until the actual leaf nodes that are not pruned are compared. The problem comes from the nature of a centroid which is an average of a collection of items (in Physics, the center of gravity). The risk is that the average may not be similar enough to the query for continued search, but specific items used to calculate the centroid may be close enough to satisfy the search. The risks of missing items and thus reducing recall increases as the standard deviation increases. Use of centroids reduces the similarity computations but could cause a decrease in recall. It should have no effect on precision since that is based upon the similarity calculations at the leaf (item) level.

In Figure 1.5 the filled circle represents the query and the filled boxes represent the centroids for the three clusters represented by the ovals. In this case, the query may only be similar enough to the end two circles for additional analysis. But there are specific items in the right cluster that are much closer to the query than the cluster centroid and could satisfy the query. These items cannot be returned because when their centroid is eliminated they are no longer considered.

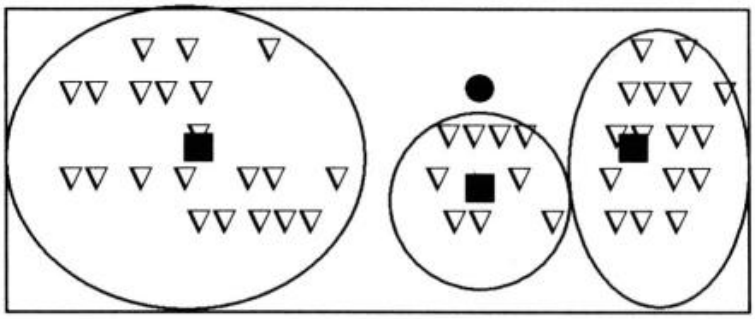


Figure 13.5 Centroid Comparisons

In their experiments they applied the clustering to the entire corpora. Although the clustering conveyed some of the content and structure of the corpora, it was shown to be less effective in retrieval than a standard similarity query (Pirolli-96). Constraining the search to the hierarchy retrieved fewer relevant items than a similarity query that focused the results on an indexed logical subset of the corpus.

13.2.2 Hidden Markov Models Techniques

Use of Hidden Markov Models for searching textual corpora has introduced a new paradigm for search. In most of the previous search techniques, the query is thought of as another "document" and the system tries to find other documents similar to it. In HMMs the documents are considered unknown statistical processes that can generate output that is equivalent to the set of queries that would consider the document relevant. Another way to look at it is by taking the general definition that a HMM is defined by output that is produced by passing some unknown key via state transitions through a noisy channel. The observed output is the query, and the unknown keys are the relevant documents. The development for a HMM approach begins with applying Bayes rule to the conditional probability:

$$P(D \text{ is } R/Q) = P(Q/D \text{ is } R) * P(D \text{ is } R) / P(Q)$$

Since we are performing the analysis from the document's perspective, the P(Q) will be the same for every document and thus can be ignored. P (D is R) is also almost an impossible task in a large diverse corpora. Relevant documents sets seem to be so sensitive to the specific queries, that trying to estimate P (D is R) does not return any noticeable improvements in query resolution. Thus the probability that a document is relevant given a specific query can be estimated by calculating the probability of the query given the document is Relevant, i.e., P (Q/D is R).

A Hidden Markov Model is defined by a set of states, a transition matrix defining the probability of moving between states, a set of output symbols and the probability of the output symbols given a particular state. The set of all possible queries is the output symbol set and the Document file defines the states. States could for example be any of the words or stems of the words in the documents. Thus the HMM process traces itself through the states of a document (e.g., the words in the document) and at each state transition has an output of query terms associated with the new state. State transitions are associated with ways that words are combined to make documents. Given the query, it is possible to calculate the probability that any particular document generated the query.

The biggest problem in using this approach is to estimate the transition probability matrix and the output (queries that could cause hits) for every document in the corpus. If there was a large training database of queries and the relevant documents they were associated with that included adequate coverage, then the problem could be solved using Estimation-Maximization algorithms (Dempster-77, Bryne-93.) But given the lack of data, Leek et. al. recommend making the transition matrix independent of specific document sets and applying simple unigram estimation for output distributions (Leek-99).

13.2.3 Ranking Algorithms

The main reason the natural language/ranking approach is more effective for end-users is that all the terms in the query are used for retrieval, with the results being ranked based on co-occurrence of query terms, as modified by statistical term-weighting. This method eliminates the often-wrong Boolean syntax used by end-users, and provides some results even if a query term is incorrect, that is, it is not the term used in the data, it is misspelled, and so on. The ranking methodology also works well for the complex queries that may be difficult for end-users to express in Boolean logic. For example, "human factors and/or system performance in medical databases" is difficult for end-users to express in Boolean logic because it contains

many high- or medium-frequency words without any clear necessary Boolean syntax. The ranking method would do well with this query.

In most of the commercial systems, heuristic rules are used to assist in the ranking of items. Generally, systems do not want to use factors that require knowledge across the corpus (e.g., inverse document frequency) as a basis for their similarity or ranking functions because it is too difficult to maintain current values as the database changes and the added complexity has not been shown to significantly improve the overall weighting process. A good example of how a commercial product integrates efficiency with theoretical concepts is the RetrievalWare system's approach to queries and ranking (RETRIEVALWARE-95).

RetrievalWare first uses indexes (inversion lists) to identify potential relevant items. It then applies coarse grain and fine grain ranking. The coarse grain ranking is based on the presence of query terms within items. In the fine grain ranking, the exact rank of the item is calculated. The coarse grain ranking is a weighted formula that can be adjusted based on completeness, contextual evidence or variety, and semantic distance. Completeness is the proportion of the number of query terms (or related terms if a query term is expanded using the RetrievalWare semantic network/thesaurus) found in the item versus the number in the query. It sets an upper limit on the rank value for the item. If weights are assigned to query terms, the weights are factored into the value. Contextual evidence occurs when related words from the semantic network are also in the item. Thus if the user has indicated that the query term "charge" has the context of "paying for an object" then finding words such as "buy," "purchase," "debt" suggests that the term "charge" in the item has the meaning the user desires and that more weight should be placed in ranking the item. Semantic distance evaluates how close the additional words are to the query term. Synonyms add additional weight; antonyms decrease weight. The coarse grain process provides an initial rank to the item based upon existence of words within the item. Since physical proximity is not considered in coarse grain ranking, the ranking value can be easily calculated.

Fine grain ranking considers the physical location of query terms and related words using factors of proximity in addition to the other three factors in coarse grain evaluation. If the related terms and query terms occur in close proximity (same sentence or paragraph) the item is judged more relevant. A factor is calculated that maximizes at adjacency and decreases as the physical separation increases. If the query terms are widely distributed throughout a long item, it is possible for the item to have a fine grain rank of zero even though it contains the query terms.

Although ranking creates a ranking score, most systems try to use other ways of indicating the rank value to the user as Hit lists are displayed. The scores have a tendency to be misleading and confusing to the user. The differences between the values may be very close or very large. It has been found to be better to indicate the general relevance of items than to be over specific.

13.3 SUMMARY

Creating the index to an Information Retrieval System defines the searchable concepts that represent the items received by a system. The user search process is the mechanism that correlates the user‘s search statement with the index via a similarity function. There are a number of techniques to define the indexes to an item. It is typically more efficient to incur system overhead at index creation time than search time. An item is processed once at index time, but there will be millions of searches against the index. Also, the user is directly affected by the response time of a search but, in general, is not aware of how long it takes from receipt of an item to its being available in the index. The selection and implementation of similarity algorithms for search must be optimized for performance and scaleable to accommodate very large databases.

It is typical during search parsing that the user’s initial search statement is expanded via a thesaurus or semantic net to account for vocabulary differences between the user and the authors. But excessive expansion takes significantly more processing and increases the response time due to the number of terms that have to be processed. Most systems have default limits on the number of new terms added to a search statement. The algorithms used as similarity measures are still in a state of evolution and are continually being modified to improve their performance. The search algorithms in a probabilistic indexing and search system are much more complex than the similarity measures described. For systems based upon natural language processing, once the initial similarity comparisons are completed, there is an additional search processing step to make use of discourse level information, adding additional precision to the final results.

13.4 KEYWORDS

Binding, Similarity Measure, Ranking,

13.5 EXERCISES

1. Discuss the sources of potential errors in the final set of search terms from when a user first identifies a need for information to the creation of the final query.
2. Why are there three levels of binding in the creation of a search?
3. Why does the numerator remain basically the same in all of the similarity measures. Discuss other possible approaches and their impact on the formulas.

13.5 REFERENCES FOR FURTHER STUDY

1. Y. Shen; D. Lun Lee, "Meta-search Method Reinforced by Cluster Descriptors", In Proceedings of the 2nd International Conference on Web Information Systems Engineering, Volume 1, p.p. 3-6, 2001.
2. List of search engines:
    a. http://thesearchenginelist.com
    b. Search: www.google.com; www.yahoo.com;
    c. www.bing.com;www.ask.com
    d. Meta search: www.dogpile.com; www.metacrawler.com;
    e. www.clusty.com; http://find.copernic.com; www.surfwax.com;
3. Directories:
    a. http://dir.yahoo.com/;
    b. www.google.com/dirhp;
    c. www.questia.com/Index.jsp www.eldis.org/;
    d. http://openlearn.open.ac.uk/; www.dmoz.org
4. Web materials:
    a. http://nlp.stanford.edu/IR-book/pdf/09expand.pdf
    b. http://orion.lcg.ufrj.br/Dr.Dobbs/books/book5/toc.htm
    c. http://en.wikipedia.org/wiki/Information_retrieval

UNIT-14: RELEVANCE FEEDBACK

Structure

14.1 INTRODUCTION TO RELEVANCE FEEDBACK

Relevance feedback is a feature of some information retrieval systems. The idea behind relevance feedback is to take the results that are initially returned from a given query and to use information about whether or not those results are relevant to perform a new query. We can usefully distinguish between three types of feedback: explicit feedback, implicit feedback, and blind or "pseudo" feedback. Explicit feedback is obtained from assessors of relevance indicating the relevance of a document retrieved for a query. This type of feedback is defined as explicit only when the assessors (or other users of a system) know that the feedback provided is interpreted as relevance judgments. Implicit feedback is inferred from user behaviour, such as noting which documents they do and do not select for viewing, the duration of time spent viewing a document, or page browsing or scrolling actions. Pseudo relevance feedback, also known as blind relevance feedback, provides a method for automatic local analysis. It automates the manual part of relevance feedback, so that the user gets improved retrieval performance without an extended interaction. The method is to do normal retrieval to find an initial set of most relevant documents, to then assume that the top "k" ranked documents are relevant, and finally to do relevance feedback as before under this assumption.

One of the major problems in finding relevant items lies in the difference in vocabulary between the authors and the user. Thesauri and semantic networks provide utility in generally expanding a user's search statement to include potential related search terms. But this still does not correlate to the vocabulary used by the authors that contributes to a particular database. There is also a significant risk that the thesaurus does not include the latest jargon being used, acronyms or proper nouns. In an interactive system, users can manually modify an inefficient query or have the system automatically expand the query via a thesaurus. The user can also use relevant items that have been found by the system (irrespective of their ranking) to improve future searches, which is the basis behind relevance feedback. Relevant items (or portions of relevant items) are used to reweight the existing query terms and possibly expand the user's search statement with new terms.

The first major work on relevance feedback was published in 1965 by Rocchio (republished in 1971: Rocchio-71). The relevance feedback concept was that the new query should be based on the old query modified to increase the weight of terms in relevant items and decrease the weight of terms that are in non-relevant items. This technique not only modified the terms in the original query but also allowed expansion of new terms from the relevant items. The formula used is:

$$Q_n = Q_o + \frac{1}{r}\sum_{i=1}^{r} DR_i - \frac{1}{nr}\sum_{j=1}^{nr} DNR_j$$

where

Qn = the revised vector for the new query
Qo = the original query
r = number of relevant items
DRi = the vectors for the relevant items
nr = number of non-relevant items
DNRj = the vectors for the non-relevant items

The factors *r* and *nr* were later modified to be constants that account for the number of items along with the importance of that particular factor in the equation. Additionally a constant was added to **Qo** to allow adjustments to the importance of the weight assigned to the original query. This led to the revised version of the formula:

$$Q_n = \alpha Q_o + \beta \sum_{i=1}^{r} DR_i - \gamma \sum_{j=1}^{nr} DNR_j$$

where $\alpha, \beta,$ and γ and are the constants associated with each factor (usually 1/*n* or 1/*nr* times a constant). The factor $\beta \sum_{i=1}^{r} DR_i$ is referred to as positive feedback because it is using the user judgments on relevant items to increase the values of terms for the next iteration of searching. The factor $\gamma \sum_{j=1}^{nr} DNR_j$ is referred to as negative feedback since it decreases the values of terms in the query vector. Positive feedback is weighted significantly greater than negative feedback. Many times only positive feedback is used in a relevance feedback environment. Positive feedback is more likely to move a query closer to a user's information needs. Negative feedback may help, but in some cases it actually reduces the effectiveness of a query. Figure 7.6 gives an example of the impacts of positive and negative feedback. The filled circles represent non-relevant items; the other circles represent relevant items. The oval represents the items that are returned from the query. The solid box is logically where the query is initially. The hollow box is the query modified by relevance feedback (positive only or negative only in the Figure).

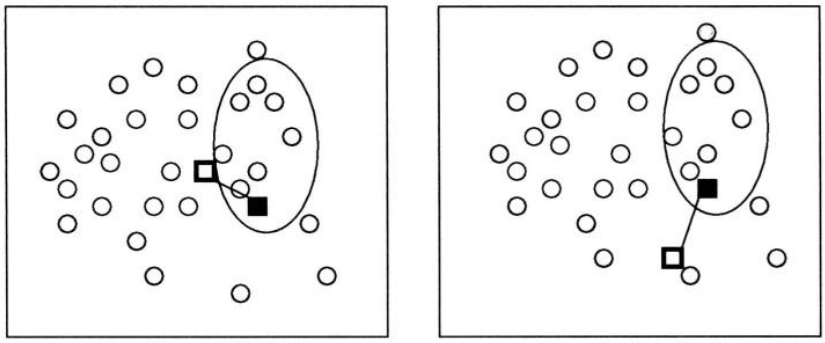


Figure 14.1 Impact of Relevance Feedback

Positive feedback moves the query to retrieve items similar to the items retrieved and thus in the direction of more relevant items. Negative feedback moves the query away from the non-relevant items retrieved, but not necessarily closer to more relevant items. Figure 1.7 shows how the formula is applied to three items (two relevant and one non-relevant). If we use the factors $\alpha = 1, \beta = ¼$ (½ times a constant ½), γ = ¼ (1/1 times a constant ¼) in the foregoing formula we get the following revised query (NOTE: negative values are changed, to a zero value in the revised Query vector):

$$Q_n = (3, 0, 0, 2, 0) + \tfrac{1}{4}(2+1, 4+3, 0+0, 0+0, 2+0) - \tfrac{1}{4}(0, 0, 4, 3, 2)$$
$$= (3\tfrac{3}{4}, 1\tfrac{3}{4}, 0\ \{-1\}, 1\tfrac{1}{4}, 0)$$

	Term 1	Term 2	Term 3	Term 4	Term 5
Q_o	3	0	0	2	0
$DOC1_r$	2	4	0	0	2
$DOC2_r$	1	3	0	0	0
$DOC3_{nr}$	0	0	4	3	3
Q_n	3¾	1¾	0	1¼	0

Figure14.2 Query Modifications via Relevance Feedback

Using the un-normalized similarity formula

$$SIM(Q_k, DOC_l) = \sum_{i=1}^{5} TERM_{k,i} * TERM_{l,I}$$

Produces the results shown in Figure 14.3:

	DOC1	DOC2	DOC3
Q_o	6	3	6
Q_n	14½	9.0	3.75

Figure 14.3 Effect of Relevance Feedback

In addition to showing the benefits of relevance feedback, this example illustrates the problems of identifying information. Although DOC3 is not relevant to the user, the initial query produced one of the highest similarity measures for it. This was caused by a query term (Term 4) of interest to the user that has a significant weight in DOC3. The fewer the number of terms in a user query, the more likely a specific term to cause non-relevant items to be returned. The modification to the query by the relevance feedback process significantly increased the similarity measure values for the two relevant items (DOC1 and DOC2) while decreasing the value of the non-relevant item. It is also of interest to note that the new query added a weight to Term 2 that was not in the original query. One reason that the user might not have initially had a value to Term 2 is that it might not have been in the user's vocabulary. For example, the user may have been searching on "PC" and "word processor" and not been aware that many authors use the specific term "Macintosh" rather than "PC."

Relevance feedback, in particular positive feedback, has been proven to be of significant value in producing better queries. Some of the early experiments on the SMART system (Ide-69, Ide-71, Salton-83) indicated the possible improvements that would be gained by the process. But the small collection sizes and evaluation techniques put into question the actual gains by using relevance feedback. One of the early problems addressed in relevance feedback is how to treat query terms that are not found in any retrieved relevant items. Just applying the algorithm would have the effect of reducing the relative weight of those terms with respect to other query terms. From the user's perspective, this may not be desired because the term may still have significant value to the user if found in the future iterations of the search process. Harper and van Rijisbergen addressed this issue in their proposed EMIM weighting scheme (Harper-78, Harper-80). Relevance feedback has become a common feature in most information systems. When the original query is modified based upon relevance feedback, the systems ensure that the original query terms are in the modified query, even if negative feedback would have eliminated them. In some systems the modified query is presented to the user to allow the user to readjust the weights and review the new terms added.

Recent experiments with relevance feedback during the TREC sessions have shown conclusively the advantages of relevance feedback. Queries using relevance feedback produce significantly better results than those being manually enhanced. When users enter queries with a few number of terms, automatic relevance feedback based upon just the rank values of items has been used. This concept in information systems called pseudo-relevance feedback, blind feedback or local context analysis (Xu-96) does not require human relevance judgments. The highest ranked items from a query are automatically assumed to be relevant and applying relevance feedback (positive only) used to create and execute an expanded query. The system returns to the user a Hit file based upon the expanded query. This technique also showed improved performance over not using the automatic relevance feedback process. In the automatic query processing tests from TREC most systems use the highest ranked hits from the first pass to generate the relevance feedback for the second pass.

14.2 SELECTIVE DISSEMINATION OF INFORMATION SEARCH

Selective Dissemination of Information, frequently called dissemination systems, are becoming more prevalent with the growth of the Internet. A dissemination system is sometimes labeled a "push" system while a search system is called a "pull" system. The differences are that in a search system the user proactively makes a decision that he needs

information and directs the query to the information system to search. In a dissemination system, the user defines a profile (similar to a stored query) and as new information is added to the system it is automatically compared to the user's profile. If it is considered a match, it is asynchronously sent to the user's "mail" file

One concept that ties together the two search statements (query and profile) is the introduction of a time parameter associated with a search statement. As long as the time is in the future, the search statement can be considered active and disseminating as items arrive. Once the time parameter is past, the user's need for the information is no longer exists except upon demand (i.e., issuing the search statement as an ad hoc query).

The differences between the two functions lie in the dynamic nature of the profiling process, the size and diversity of the search statements and number of simultaneous searches per item. In the search system, an existing database exists. As such, corpora statistics exist on term frequency within and between terms. These can be used for weighting factors in the indexing process and the similarity comparison (e.g., inverse document frequency algorithms). A dissemination system does not necessarily have a retrospective database associated with it. Thus its algorithms need to avoid dependency upon previous data or develop a technique to estimate terms for their formula. This class of system is also discussed as a binary classification system because there is no possibility for real time feedback from the user to assist in search statement refinement. The system makes a binary decision to reject or file the item (Lewis-95).

Profiles are relatively static search statements that cover a diversity of topics. Rather than specifying a particular information need, they usually generalize all of the potential information needs of a user. They are focused on current information needs of the user. Thus profiles have a tendency to contain significantly more terms than an ad hoc query (hundreds of terms versus a small number). The size tends to make them more complex and discourages users from wanting to change them without expert advice.

One of the first commercial search techniques for dissemination was the Logicon Message Dissemination System (LMDS). The system originated from a system created by Chase, Rosen and Wallace (CRW Inc.). It was designed for speed to support the search of thousands of profiles with items arriving every 20 seconds. It demonstrated one approach to the problem where the profiles were treated as the static database and the new item acted like the query. It uses the terms in the item to search the profile structure to identify those profiles whose logic

could be satisfied by the item. The system uses a least frequently occurring tri-graph (three characters) algorithm that quickly identifies which profiles are not satisfied by the item. The potential profiles are analyzed in detail to confirm if the item is a hit.

Another example of a dissemination approach is the Personal Library Software (PLS) system. It uses the approach of accumulating newly received items into the database and periodically running user's profiles against the database. This makes maximum use of the retrospective search software but loses near real time delivery of items. More recent examples of a similar approach are the Retrievalware and the InRoute software systems. In these systems the item is processed into the searchable form. Since the Profiles are relatively static, some use is made in identifying all the terms used in all the profiles. Any words in the items that are members of this list cannot contribute to the similarity process and thus are eliminated from the search structure. Every profile is then compared to the item. Retrieval ware uses a statistical algorithm but it does not include any corpora data. Thus not having a database does not affect its similarity measure.

In Route, like the 1NQUERY system used against retrospective database, uses inverse document frequency information. It creates this information as it processes items, storing and modifying it for use as future items arrive. This would suggest that the values would be continually changing as items arrive until sufficient items have arrived to stabilize the inverse document frequency weights. Relevance feedback has been proven to enhance the search capabilities of ad hoc queries against retrospective databases. Relevance feedback can also be applied to dissemination systems. Unlike an ad hoc query situation, the dissemination process is continuous, and the issue is the practicality of archiving all of the previous relevance judgments to be used in the relevance feedback process.

Another approach to dissemination uses a statistical classification technique and explicit error minimization to determine the decision criteria for selecting items for a particular profile (Schutze-95). In this case, the classification process is related to assignment for each item into one of two classes: relevant to a user's profile or non-relevant. Error minimization encounters problems in high dimension spaces. The dimensionality of an information space is defined by the number of unique terms where each term is another dimension. This is caused by there being too many dimensions for a realistic training set to establish the error minimization parameters. To reduce the dimensionality, a version of latent semantic indexing (LSI) can be used. The process requires a training data set along with its associated profiles. Relevance feedback is an example of a simple case of a learning algorithm that does not use

error minimization. Other examples of algorithms used in linear classifiers that perform explicit error minimization are linear discriminant analysis, logistic regression and linear neural networks.

Schutze et al. used two approaches to reduce the dimensionality: selecting a set of existing features to use or creating a new much smaller set of features that the original features are mapped into. A χ^2 measure was used to determine the most important features. The test was applied to a table that contained the number of relevant (N_r) and non-relevant items in which a term occurs plus the number of relevant and non-relevant (N_{nr}) items in which the term does not occur $(N_{r-}, N_{nr-}$ respectively). The formula used was:

$$\chi^2 = \frac{N(N_r N_{nr-} - N_{r-} N_{nr})^2}{(N_r + N_{r-})(N_{nr} + N_{nr-})(N_r + N_{nr})(N_{r-} + N_{nr-})}$$

To focus the analysis, only items in the local region defined by a profile were analyzed. The chi-squared technique provides a more effective mechanism than frequency of occurrence of terms. A high χ^2 score indicates a feature whose frequency has a significant dependence on occurrence in a relevant or non-relevant item.

An alternative technique to identify the reduced feature (vector) set is to use a modified latent semantic index (LSI) technique to determine a new reduced set of concept vectors. The technique varies from the LSI technique by creating a separate representation of terms and items by each profile to create the "local" space of items likely to be relevant (i.e., Local LSI). The results of the analysis go into a learning algorithm associated with the classification technique (Hull-94). The use of the profile to define a local region is essential when working with large databases. Otherwise the number of LSI factors is in the hundreds and the ability to process them is currently unrealistic. Rather than keeping the LSI factors separate per profile, another approach is to merge the results from all of the queries into a single LSI analysis (Dumais-93). This increases the number of factors with associated increase in computational complexity.

Once the reduced vector set has been identified, then learning algorithms can be used for the classification process. Linear discriminate analysis, logistic regression and neural networks are three possible techniques that were compared by Schutze et al. Other possible techniques are classification trees (Tong-94, Lewis-94a), Bayesian networks (Croft-94), Bayesian

classifiers (Lewis-92), rules induction (Apte-94), nearest neighbor techniques (Masand-92, Yang-94), and least square methods (Fuhr-89). Linear discrimination analysis uses the covariance class for each document class to detect feature dependence (Gnanadesikan-79).

Assuming a sample of data from two groups with n1 and n2 members, mean vectors and $\bar{x}_1$ and $\bar{x}_2$ covariance matrices C1 and C2 respectively, the objective is to maximize the separation between the two groups. This can be achieved by maximizing the distance between the vector means, scaling to reflect the structure in the pooled covariance matrix. Thus choose *a* such that:

$$a^* = \arg_a \max \frac{a^T(\bar{x}_1 - \bar{x}_2)}{\sqrt{a^T C_a}}$$

is maximized where T is the transpose and $(n_1 + n_2 - 2)C = (n_1 - 1)C_1 + (n_2 - 1)C_2$.

Since C is positive, the Cholesky decomposition of $C = R^T$. Let then $b = Ra$; the formula becomes;

$$a^* = \arg_b \max \frac{b^T R^{T-1}(\bar{x}_1 - \bar{x}_2)}{\sqrt{b^T b}}$$

which is maximized by choosing $b \propto R^{T-1}(\bar{x}_1 - \bar{x}_2)$. This means:

$$a^* = R^{-1}b = C^{-1}(\bar{x}_1 - \bar{x}_2)$$

The one dimensional space defined by $y = a^{*T}x$ should cause the group means to be well separated. To produce a non-linear classifier, a pair of shrinkage parameters is used to create a very general family of estimators for the group covariance matrix (Freidman-89). This process called Regularized Discriminant Analysis looks at a weighted combination of the pooled and unpooled covariance matrices. The optimal values of the shrinkage parameters are selected based upon the cross validation over the training set. The non-linear classifier produced by this technique has not been shown to make major improvements in the classification process (Hull-95).

A second approach is to use logistic regression (Cooper-94a). It models a binary response variable by a linear combination of one or more predictor variables, using a logit link function:

$$g(\pi) = \log(\pi/(1 - \pi))$$

and modelling variance with a binomial random variable. This is achieved by modeling the dependent variable $\log(\pi/(1 - \pi))$ as a linear combination of independent variables using a form $g(\pi) = x_i\beta$. In this formula π is the estimated response probability (probability of relevance), **xi** is the feature vector (reduced vector) for document *I*, and **β** is the weight vector which is estimated from the matrix of feature vectors. The optimal value **β** of can be calculated using the maximum likelihood and the Newton-Raphson method of numerical optimization (McCullagh-89). The major difference from previous experiments using logistic regression is that Schutze et al. do not use information from all the profiles but restrict the analysis for each profile.

A third technique is to use neural networks for the learning function. A neural network is a network of input and output cells (based upon neuron functions in the brain) originating with the work of McCulloch and Pitts (McCulloch-43). Each input pattern is propagated forward through the network. When an error is detected it is propagated backward adjusting the cell parameters to reduce the error, thus achieving learning. This technique is very flexible and can accommodate a wide range of distributions. A major risk of neural networks is that they can overfit by learning the characteristics of the training data set and not be generalized enough for the normal input of items. In applying training to a neural network approach, a validation set of items is used in addition to the training items to ensure that overfitting has not occurred. As each iteration of parameter adjustment occurs on the training set, the validation set is retested. Whenever the errors on the validation set increase, it indicates that overfitting is occurring and establishes the number of iterations on training that improve the parameter values while not harming generalization.

The linear and non-linear architectures for an implementation of neural nets is shown in Figure 14.4.

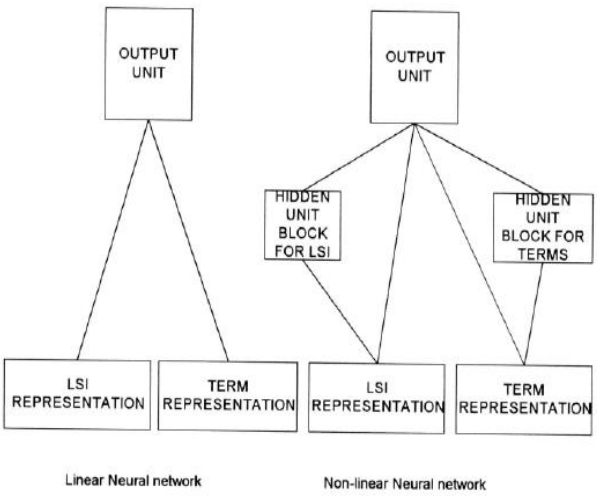


Figure 14.4 Linear and Non-linear networks

In the non-linear network, each of the hidden blocks consists of three hidden units. A hidden unit can be interpreted as feature detectors that estimate the probability of a feature being present in the input. Propagating this to the output unit can improve the overall estimation of relevance in the output unit. The networks show input of both terms and the LSI representation (reduced feature set). In both architectures, all input units are directly connected to the output units. Relevance is computed by setting the activations of the input units to the document's representation and propagating the activation through the network to the output unit, then propagating the error back through the network using a gradient descent algorithm (Rumelhart-95). A sigmoid was chosen as:

$$f(x) = \frac{e^x}{1+e^x}$$

as the activation function for the units of the network (Schutze-95). In this case backpropagation minimizes the same error as logistic regression (Rumelhart-95a), The cross-entropy error is:

$$L = -\sum t_i \log \sigma_i + 1 - t_i) \log(1 - \sigma_i)$$

where is the relevance for document *I* and is the estimated relevance (or activation of the output unit) for document *i.* The definition of the sigmoid is equivalent to:

$$x = \log \left(\frac{f(x)}{1-f(x)}\right)$$

which is the same as the log link function.

Schutze et al. performed experiments with the Tipster test database to compare the three algorithms. They show that the linear classification schemes perform 10-15 per cent better than the traditional relevance feedback. To use the learning algorithms based upon error minimization and numerical computation one must use some technique of dimensionality reduction. Their experiments show that local latent semantic indexing is best for linear discrimination analysis and logistic regression since they have no mechanism for protecting against over fitting. When there are mechanisms to avoid over fitting such as in neural networks, other less precise techniques of dimension reduction can be used. This work suggests that there are alternatives to the statistical classification scheme associated with profiles and dissemination.

An issue with Mail files is the logical reorganization associated with display of items. In a retrospective query, the search is issued once and the hit list is a static file that does not change in size or order of presentation. The dissemination function is always adding items that satisfy a user's profile to the user's Mail file. If the items are stored sorted by rank, then the relative order of items can always be changing as new items are inserted in their position based upon the rank value. This constant reordering can be confusing to the user who remembers items by spatial relationships as well as naming. Thus the user may remember an item next to another item is of significant interest. But in trying to retrieve it at a later time, the reordering process can make it significantly harder to find.

14.3 WEIGHTED SEARCHES OF BOOLEAN SYSTEMS

The two major approaches to generating queries are Boolean and natural language. Natural language queries are easily represented within statistical models and are usable by the similarity measures discussed. Issues arise when Boolean queries are associated with weighted index systems. Some of the issues are associated with how the logic (AND, OR, NOT) operators function with weighted values and how weights are associated with the query terms. If the operators are interpreted in their normal interpretation, thay act too restrictive or too general (i.e., AND and OR operators respectively). Salton, Fox and Wu showed that using the strict definition of the operators will suboptimize the retrieval expected by the user (Salton-83a). Closely related to the strict definition problem is the lack of ranking that is missing from a pure Boolean process. Some of the early work addressing this problem recognized the fuzziness associated with mixing Boolean and weighted systems (Brookstein-78, Brookstein-80).

To integrate the Boolean and weighted systems model, Fox and Sharat proposed a fuzzy set approach (Fox-86). Fuzzy sets introduce the concept of degree of membership to a set (Zadeh-65). The degree of membership for AND and OR operations are defined as:

$$DEG_{A \cap B} = \min(DEG_A, DEG_B)$$

$$DEG_{A \cup B} = \max(DEG_A, DEG_B)$$

where A and B are terms in an item. DEG is the degree of membership. The Mixed Min and Max (MMM) model considers the similarity between query and document to be a linear combination of the minimum and maximum item weights. Fox proposed the following similarity formula:

$$SIM(QUERY_{OR}, DOC) = C_{OR\,1} * \max(DOC1_1, DOC_2, \ldots, DOC_n) + C_{OR2} * \min(DOC_1, DOC_2, \ldots, DOC_n)$$

$$SIM(QUERY_{AND}, DOC) = C_{AND1} * \min(DOC_1, DOC_2, \ldots, DOC_n) + C_{AND2} * \max(DOC1_1, DOC_2, \ldots, DOC_n)$$

where **COR1**and **COR2** are weighting coefficients for the OR operation and **CAND1** and **CAND2** are the weighting coefficients for the AND operation. Lee and Fox found in their experiments that the best performance comes when **CAND1** is between 0.5 to 0.8 and is greater than 0.2.

The MMM technique was expanded by Paice (Paice-84) considering all item weights versus the maximum/minimum approach. The similarity measure is calculated as:

$$SIM(QUERY\ DOC) = \sum_{i=1}^{n} r^{i-1} d_i \ / \ \sum_{i=1}^{n} r^{i-1}$$

where the d_i's are inspected in ascending order for AND queries and descending order for OR queries. The *r* terms are weighting coefficients. Lee and Fox showed that the best values for *r* are 1.0 for AND queries and 0.7 for OR queries (Lee-88). This technique requires more computation since the values need to be stored in ascending or descending order and thus must be sorted.

An alternative approach is using the P-norm model which allows terms within the query to have weights in addition to the terms in the items. Similar to the Cosine similarity technique, it considers the membership values $(d_{A1}, \ldots, d_{An})$ to be coordinates in an "n" dimensional space. For an OR query, the origin (all values equal zero) is the worst possibility. For an

AND query the ideal point is the unit vector where all the D_i values equal 1. Thus the best ranked documents will have maximum distance from the origin in an OR query and minimal distance from the unit vector point. The generalized queries are:

$$Q_{OR} = (A_1,a_1) \text{ OR } (A_2,a_2) \text{ OR } \ldots \text{ OR } (A_n,a_n)$$

$$Q_{AND} = (A_1,a_1) \text{ AND } (A_2,a_2) \text{ AND } \ldots \text{ AND } (A_n,a_n)$$

The operators (AND and OR) will have a strictness value assigned that varies from 1 to infinity where infinity is the strict definition of the Boolean operator. The values are the query term weights. If we assign the strictness value to a parameter labeled "S" then the similarity formulas between queries and items are:

$$SIM(Q_{OR},DOC) = \sqrt[S]{(a_1^S d_{A1}^S + \bullet\bullet\bullet + a_n^S d_{An}^S)/(a_1^S + a_2^S + \bullet\bullet\bullet + a_n^S)}$$

$$SIM(Q_{AND},DOC) = 1 - \sqrt[S]{(a_1^S(1-d_{A1})^S + \bullet\bullet\bullet + a_n^S(1-d_{An})^S)/(a_1^S + a_2^S + \bullet\bullet\bullet + a_n^S)}$$

$$SIM(Q_{not},DOC) = 1 - SIM(Q, DOC)$$

Another approach suggested by Salton provides additional insight into the issues of merging the Boolean queries and weighted query terms under the assumption that there are no weights available in the indexes (Salton-83). The objective is to perform the normal Boolean operations and then refine the results using weighting techniques. The following procedure is a modification to his approach for defining search results. The normal Boolean operations produce the following results:

"A OR B" retrieves those items that contain the term A or the term B or both

"A AND B" retrieves those items that contain both terms A and B

"A NOT B" retrieves those items that contain term A and not contain term B.

If weights are then assigned to the terms between the values 0.0 to 1.0, they may be interpreted as the significance that users are placing on each term. The value 1.0 is assumed to be the strict interpretation of a Boolean query. The value 0.0 is interpreted to mean that the user places little value on the term. Under these assumptions, a term assigned a value of 0.0 should have no effect on the retrieved set. Thus

"A1 OR B0" should return the set of items that contain A as a term

"A1 AND B0" will also return the set of items that contain term A

"A1 NOT B0" also return set A.

This suggests that as the weight for term B goes from 0.0 to 1.0 the resultant set changes from the set of all items that contains term A to the set normally generated from the Boolean operation. The process can be visualized by use of the VENN diagrams shown in Figure 7.10. Under the strict interpretation A1 OR B1 would include all items that are in all the areas in the VENN diagram. A1 OR B0 would be only those items in A (i.e., the white and black dotted areas) which is everything except items in "B NOT A" (the grey area.) Thus as the value of query term B goes from 0.0 to 1.0, items from "B NOT A" are proportionally added until at 1.0 all of the items will be added.

Similarly, under the strict interpretation A1 AND B1 would include all of the items that are in the black dotted area. A1 AND B0 will be all of the items in A as described above. Thus, as the value of query term B goes from 1.0 to 0.0 items will be proportionally added from "A NOT B" (white area) until at 0.0 all of the items will be added.

Finally, the strict interpretation of A1 NOT B1 is grey area while A1 NOT B0 is all of A. Thus as the value of B goes from 0.0 to 1.0, items are proportionally added from "A AND B" (black dotted area) until at 1.0 all of the items have been added.

The final issue is the determination of which items are to be added or dropped in interpreting the weighted values. Inspecting the items in the totally strict case (both terms having weight 1.0) and the case where the value is 0.0 there is a set of items that are in both solutions (invariant set).

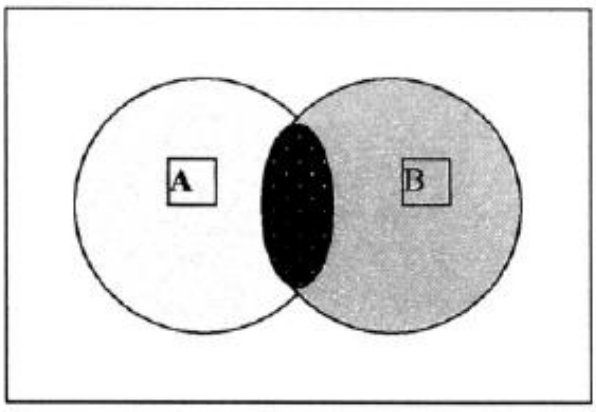


Figure 14.5 Venn diagram

In adding items they should be the items most similar to the set of items that do not change in either situation. In dropping items, they should be the items least similar to those that are in both situations.

Thus the algorithm follows the following steps:

1. Determine the items that are satisfied by applying strict interpretation of the Boolean functions
2. Determine the items that are part of the set that is invariant
3. Determine the Centroid of the invariant set
4. Determine the number of items to be added or deleted by multiplying the term weight times the number of items outside of the invariant set and rounding up to the nearest whole number.
5. Determine the similarity between items outside of the invariant set and the Centroid
6. Select the items to be included or removed from the final set.

Figure 14.6 gives an example of solving a weighted Boolean query.

$QUERY_1$ ends up with a set containing all of the items that contain the term "Computer" and two items from the set "computer" NOT "program." The symbol $\lceil\ \rceil$ stands for rounding up to the next integer. In $QUERY_2$ the final set

	Computer	program	cost	sale
D1	0	4	0	8
D2	0	2	0	0
D3	4	0	2	4
D4	0	6	4	6
D5	0	4	6	4
D6	6	0	4	0
D7	0	0	0	0
D8	4	2	0	2

$Q1 = QUERY_1 = Computer_{1.0}$ OR $program_{.333}$
$Q2 = QUERY_2 = cost_{.75}$ AND $sale_{1.0}$

$Q1_{strict\ interpretation}$ = (D1, D2, D3, D4, D5, D6, D8)
$Q2_{strict\ interpretation}$ = (D3, D4, D5)

$Q1_{invariant}$ = (D8)
$Q2_{invariant}$ = (D3, D4, D5)

$Q1_{optional}$ = (D1, D2, D3, D4, D5, D6) thus $\lceil$.333 times 6 items$\rceil$ = 2 items
$Q2_{optonal}$ = (D1, D8) which means $\lceil$ (1 - .75) times 2 items$\rceil$ = 1 item

Figure 14.6 Example of Weighted Boolean Query contains all of set "cost" AND "sale" plus .25 of the set of "sale" NOT "cost." Using the simple similarity measure:

$$SIM(Item_i, Item_j) = \Sigma (Term_{i,k}) (Term_{j,k})$$

leads to the following set of similarity values based upon the centroids:

CENTROID (Q1) = (D8) = (4,2,0,2)

CENTROID (Q2) = (D3, D4, D5) = 1/3(4+0+0, 0+6+4, 2+4+6, 4+6+4)

SIM(CENTROIDQ1,D1)= (0+8+0+16) = 24

SIM(CENTROIDQ1,D2)= (0+4+0+0) = 4

SIM(CENTROIDQ1,D3)= (16+0+0+8) = 24

SIM(CENTROIDQ1,D4)= (0+12+0+12) = 24

SIM(CENTROIDQ1,D5)= (0+8+0+8) = 16

SIM(CENTROIDQ1,D6)= (24+0+0+0) = 24

SIM(CENTROIDQ2,D7)= 1/3(0+40+0+112) = 1/3(152)

SIM(CENTROIDQ2,D8)= 1/3(16+20+0+28) = 1/3(64)

For Q1, two additional items are added to the invariant set $(D8) \cup (D1, D3)$,by choosing the lowest number items because of the tie at 24, giving the answer of (D1, D3, D8). For Q2, one additional item is added to the invariant set (D3, D4, $D5) \cup (D1)$ giving the answer (D1, D3, D4, D5).

14.4 SEARCHING THE INTERNET AND HYPERTEXT

The Internet has multiple different mechanisms that are the basis for search of items. The primary techniques are associated with servers on the Internet that create indexes of items on the Internet and allow search of them. Some of the most commonly used nodes are YAHOO, AltaVista and Lycos. In all of these systems there are active processes that visit a large number of Internet sites and retrieve textual data which they index. The primary design decisions are on the level to which they retrieve data and their general philosophy on user access. LYCOS (http://www.lycos.com) and AltaVista automatically go out to other Internet

sites and return the text at the sites for automatic indexing (http://www.altavista.digital.com). Lycos returns home pages from each site for automatic indexing while Altavista indexes all of the text at a site. The retrieved text is then used to create an index to the source items storing the Universal Resource Locator (URL) to provide to the user to retrieve an item. All of the systems use some form of ranking algorithm to assist in display of the retrieved items. The algorithm is kept relatively simple using statistical information on the occurrence of words within the retrieved text.

Closely associated with the creation of the indexes is the technique for accessing nodes on the Internet to locate text to be indexed. This search process is also directly available to users via Intelligent Agents. Intelligent Agents provide the capability for a user to specify an information need which will be used by the Intelligent Agent as it independently moves between Internet sites locating information of interest. There are six key characteristics of intelligent agents (Heilmann-96):

1. Autonomy - the search agent must be able to operate without interaction with a human agent. It must have control over its own internal states and make independent decisions. This implies a search capability to traverse information sites based upon pre-established criteria collecting potentially relevant information.
2. Communications Ability - the agent must be able to communicate with the information sites as it traverses them. This implies a universally accepted language defining the external interfaces (e.g., Z39.50).
3. Capacity for Cooperation - this concept suggests that intelligent agents need to cooperate to perform mutually beneficial tasks.
4. Capacity for Reasoning - There are three types of reasoning scenarios (Roseler-94):
    a. Rule-based - where user has defined a set of conditions and actions to be taken
    b. Knowledge-based - where the intelligent agents have stored previous conditions and actions taken which are used to deduce future actions.
    c. Artificial evolution based - where intelligent agents spawn new agents with higher logic capability to perform its objectives.
5. Adaptive Behaviour - closely tied to 1 and 4 , adaptive behaviour permits the intelligent agent to assess its current state and make decisions on the actions it should take

6. Trustworthiness - the user must trust that the intelligent agent will act on the user's behalf to locate information that the user has access to and is relevant to the user.

There are many implementation aspects of Intelligent Agents. They include communications to traverse the Internet, how to wrap the agent in an appropriate interface shell to work within an Internet server, and security and protection for both the agent and the servers. Although these are critical for the implementation of the agents, the major focus for information storage and retrieval is how to optimize the location of relevant items as the agent performs its task. This requires expansion of search capabilities into conditional and learning feedback mechanisms that are becoming major topics in information retrieval.

Automatic relevance feedback is being used in a two-step process to enhance user's queries to include corpora-specific terminology. As an intelligent agent moves from site to site, it is necessary for it to use similar techniques to learn the language of the authors and correlate it to the search need of the user. How much information gained from relevance feedback from one site should be carried to the next site has yet to be resolved. Some basic groundwork is being laid by the work on incremental relevance feedback discussed earlier. It will also need capabilities to normalize ranking values across multiple systems. The quantity of possible information being returned necessitates a merged ranking to allow the user to focus on the most likely relevant items first.

Finally, there is the process of searching for information on the Internet by following Hyperlinks. A Hyperlink is an embedded link to another item that can be instantiated by clicking on the item reference. Frequently hidden to the user is a URL associated with the text being displayed. As discussed in Chapter 5, inserting hyperlinks in an item is a method of indexing related information. One of the issues of the existing Hyperlink process is the inability for the link to have attributes. In particular, a link may be a pointer to another object that is an integral aspect of the item being displayed (e.g. an embedded image or quoted text in another item). But the reference could also be to another item that generally supports the current text. It could also be to another related topic that the author feels may be of interest to the reader. There are many other interpretations of the rationale behind the link that are author specific.

Understanding the context of the link in the item being viewed determines the utility of following the associated path. Thus the Hyperlinks create a static network of linked items based upon an item being viewed. The user can manually move through this network space by following links. The search in this sense is the ability to start with an item and create the

network of associated items (i.e., following the links). The results of the search is a network diagram that defines the interrelated items which can be displayed to the user to assist in identification of where the user is in the network and to facilitate movement to other nodes (items) within the network (Gershon-95, Hasan-95, Mukherjea-95, Munzner-95). The information retrieval aspect of this problem is how to automatically follow the hyperlinks and how the additional information as each link is instantiated impacts the resolution of the user's search need. One approach is to assign the weights to the terms in original and linked items to use with the search statement to determine hits.

New search capabilities are continually becoming available on the Internet. Dissemination systems are proliferating to provide individual users with items they are potentially interested in for personal or business reasons. Some examples are the Pointcast system, FishWrap newspaper service at MIT and SFGATE (San Francisco Examiner and San Francisco Chronicle) that allow users to define specific areas of interest. Items will be e-mailed as found or stored in a file for later retrieval. The systems will continually update your screen if you are on the Internet with new items as they are found (http://fishwrapdocs. www.media.mit.edu/docs/, http:/www.sfgate.com, http:/www.pointcast.com). There are also many search sites that collect relevance information from user interaction and use relevance feedback algorithms and proprietary heuristics and provide modifications on information being delivered. Firefly interacts with a user, learning the user's preferences for record albums and movies. It provides recommendations on potential products of interest.

The Firefly system also compares the user's continually changing interest profile with other users and informs users of others with similar interests for possible collaboration (http:/www.ffly.com). Another system that uses feedback across multiple users to categorize and classify interests is the Empirical Media system (http:/www.empiracal.com). Based upon an individual user's relevance ranking of what is being displayed the system learns a user's preference. It also judges from other user's rankings of items the likelihood that an item will be of interest to other users that show the same pattern of interest. Thus it uses this "Collaborative Intelligence" in addition to its internal ranking algorithms to provide a final ranking of items to individual users. Early research attempts at using queries across multiple users to classify document systems did not show much promise (Salton-83). But the orders of magnitude increase (million times greater or more) in user interaction from the Internet provides a basis for realistic clustering and earning.

14.5 KEYWORDS

Relevance feedback, Selective Dissemination, Internet, Hypertext, Boolean system, Sum of products, cosine rule, Hirachical clustering, Centroid, Hidden Markov Model, Nonlinear Linear network, Neural Network, weighted search

14.6 SUMMARY

Relevance feedback is an alternative to thesaurus expansion to assist the user in creating a search statement that will return the needed information. Thesaurus and semantic net expansions are dependent upon the user's ability to use the appropriate vocabulary in the search statement that represents the required information. If the user selects poor terms, they will be expanded with many more poor terms. Thesaurus expansion does not introduce new concepts that are relevant to the users information need, it just expands the description of existing concepts. Relevance feedback starts with the text of an item that the user has identified as meeting his information need; incorporating it into a revised search statement. The vocabulary in the relevant item text has the potential for introducing new concepts that better reflect the user's information need along with adding additional terms related to existing search terms and adjusting the weights (importance) of existing terms.

Selective Dissemination of Information search is different from searches against the persistent information database in that it is assumed there is no information from a large corpus available to determine parameters in determining a temporary index for the item to use in the similarity comparison process (e.g., inverse document frequency factors.) An aspect of dissemination systems that helps in the search process is the tendency for the profiles to have significantly more terms than ad hoc queries. The additional information helps to identify relevant items and increase the precision of the search process. Relevance feedback can also be used with profiles with some constraints. Relevance feedback used with ad hoc queries against an existing database tends to move the terminology defining the search concepts towards the information need of the user that is available in the current database. Concepts in the initial search statement will eventually lose importance in the revised queries if they are not in the database. The goal of profiles is to define the coverage of concepts that the user cares about if they are ever found in new items. Relevance feedback applied to profiles aides the user by enhancing the search profile with new terminology about areas of interest. But, even though a concept has not been found in any items received, that area may still be of

critical importance to the user if it ever is found in any new items. Thus weighting of original terms takes on added significance over the ad hoc situation.

Searching the Internet for information has brought into focus the deficiencies in the search algorithms developed to date. The ad hoc queries are extremely short (usually less than three terms) and most users do not know how to use the advanced features associated with most search sites. Until recently research had focused on a larger more sophisticated query. With the Internet being the largest most available information system supporting information retrieval search, algorithms are in the process of being modified to account for the lack of information provided by the users in their queries. Intelligent Agents are being proposed as a potential mechanism to assist users in locating the information they require. The requirements for autonomy and the need for reasoning in the agents will lead to the merging of information retrieval algorithms and the learning processes associated with Artificial Intelligence. The use of hyperlinks is adding another level of ambiguity in what should be defined as an item. When similarity measures are being applied to identify the relevance weight, how much of the hyperlinked information should be considered part of the item? The impacts on the definition of information retrieval boundaries are just starting to be analyzed while experimental products are being developed in Web years and immediately being made available.

14.7 QUESTIONS

4. Given the following set of retrieved documents with relevance judgments

TERM	T1	T2	T3	T4	T5	T6
QUERY	0	0	4	2	6	0
REL D1	0	4	4	0	2	0
REL D2	0	2	6	0	1	0
NOT REL D3	6	0	0	6	1	0
NOT REL D4	4	0	1	2	0	10

   a. Calculate a new query using a factor of 1/2 for positive feedback and ¼ for negative feedback
   b. Determine which documents would be retrieved by the original and by the new query
   c. Discuss the differences in documents retrieved by the original versus the new query.

5. Is the use of positive feedback always better than using negative feedback to improve a query?
6. What are some potential ambiguities in use of relevance feedback on hypertext documents?

14.8 REFERENCES FOR FURTHER READINGS

C. Manning, P. Raghavan and H. Sch¨utze, Introduction to Information Retrieval

Chris Buckley and Gerard Salton and James Allan The effect of adding relevance information in a relevance feedback environment (1994)

Ian Ruthven and Mounia Lalmas A survey on the use of relevance feedback for information access systems (2003)

Jimmy Lin - Lecture notes on Relevance feedback - adapted from Doug Oard's

Jinxi Xu and W. Bruce Croft, Query expansion using local and global document analysis, in Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR), 1996.

Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. Information Retrieval: Implementing and Evaluating Search Engines. MIT Press, Cambridge, Mass., 2010.

Yuanhua Lv and ChengXiang Zhai, Positional relevance model for pseudo-relevance feedback, in Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval (SIGIR), 2010.

http://citeseer.ist.psu.edu/buckley94effect.html

http://nlp.stanford.edu/IR-book/pdf/chapter09-queryexpansion.pdf

http://personal.cis.strath.ac.uk/ir/papers/ker.pdf

UNIT 15: INFORMATION VISUALIZATION

Structure

15.1 INTRODUCTION TO INFORMATION VISUALIZATION

Information visualisation is the visual presentation of abstract information spaces and structures to facilitate their rapid assimilation and understanding. More specifically, visualization should make large datasets coherent (present huge amounts of information compactly), present information from various viewpoints, present information at several levels of detail (from overviews to fine structure) and support visual comparisons.

The primary focus on Information Retrieval Systems has been in the areas of indexing, searching and clustering versus information display. This has been due to the inability of technology to provide the technical platforms needed for sophisticated display; academic's focusing on the more interesting algorithmic based search aspects of information retrieval, and the multi-disciplinary nature of the human-computer interface. System designers need to treat the display of data as visual computing instead of treating the monitor as a replica of paper.

Functions that is available with electronic display and visualization of data that were

- modify representations of data and information or the display condition (e.g., changing color scales)
- use the same representation while showing changes in data (e.g., moving between clusters of items showing new linkages)
- animate the display to show changes in space and time

- Enable interactive input from the user to allow dynamic movement between information spaces and allow the user to modify data presentation to optimize personal preferences for understanding the data.
- Create hyperlinks under user control to establish relationships between data.

Information Visualization addresses how the results of a search may be optimally displayed to the users to facilitate their understanding of what the search has provided and their selection of most likely items of interest to read. Visual displays can consolidate the search results into a form easily processed by the user's cognitive abilities, but in general they do not answer the specific retrieval needs of the user other than suggesting database coverage of the concept and related concepts.

The theoretical disciplines of cognitive engineering and perception provide a theoretical base for information visualization. Cognitive engineering derives design principles for visualization techniques from what we know about the neural processes involved with attention, memory, imagery and information processing of the human visual system. Thus, the visual representation of an item plays as important a role as its symbolic definition in cognition. Cognitive engineering results can be applied to methods of reviewing the concepts contained in items selected by search of an information system.

Visualization can be divided into two broad classes: link visualization and attribute (concept) visualization. Link visualization displays relationships among items. Attribute visualization reveals content relationships across large numbers of items. Related to attribute visualization is the capability to provide visual cues on how search terms affected the search results. This assists a user in determining changes required to search statements that will return more relevant items.

Background: The beginnings of the theory of visualization began over 2400 years ago. The philosopher Plato discerned that we perceive objects through the senses, using the mind. Our perception of the real world is a translation from physical energy from our environment into encoded neural signals. The mind is continually interpreting and categorizing our perception of our surroundings. Use of a computer is another source of input to the mind's processing functions. Text-only interfaces reduce the complexity of the interface but also restrict use of the more powerful information processing functions the mind has developed since birth.

Information visualization is a relatively new discipline growing out of the debates in the 1970s on the way the brain processes and uses mental images. It required significant advancements in technology and information retrieval techniques to become a possibility. One of the earliest researches in information visualization, discussed the concept of "semantic road maps" that could provide a user a view of the whole database. The road maps show the items that are related to a specific semantic theme. The user could use this view to focus his query on a specific semantic portion of the database. The concept was extended in the late 1960s, emphasizing a spatial organization those maps to the information in the database. a non-linear mapping algorithm that could reveal document associations providing the information required to create a road map or spatial organization was implemented.

In the 1990s technical advancements along with exponential growth of available information moved the discipline into practical research and commercialization. Information visualization techniques have the potential to significantly enhance the user's ability to minimize resources expended to locate needed information. The way users interact with computers changed with the introduction of user interfaces based upon Windows, Icons, Menus, and Pointing devices. Although movement in the right direction to provide a more natural human interface, the technologies still required humans to perform activities optimized for the computer to understand. [Rose96]

Introduction: Although using text to present an overview of a significant amount of information makes it difficult for the user to understand the information, it is essential in presenting the details. In information retrieval, the process of getting to the relevant details starts with filtering many items via a search process. The results of this process are still a large number of potentially relevant items. In most systems the results of the search are presented as a textual list of each item perhaps ordered by rank. The user has to read all of the pages of lists of the items to see what is in the Hit list. Understanding the human cognitive process associated with visual data suggests alternative ways of presenting and manipulating information to focus on the likely relevant items. There are many areas that information visualization and presentation can help the user:

- reduce the amount of time to understand the results of a search and likely clusters of relevant information
- yield information that comes from the relationships between items versus treating each item as independent
- perform simple actions that produce sophisticated information search functions

The exponential growth in available information produces large Hit files from most searches. To understand issues with the search statement and retrieved items, the user has to review a significant number of status screens. Even with the review, it is hard to generalize if the search can be improved. Information visualization provides an intuitive interface to the user to aggregate the results of the search into a display that provides a high-level summary and facilitates focusing on likely centres of relevant items. The query logically extracts a virtual workspace (information space) of potential relevant items which can be viewed and manipulated by the user. By representing the aggregate semantics of the workspace, relationships between items become visible. It is impossible for the user to perceive these relationships by viewing the items individually. The aggregate presentation allows the user to manipulate the aggregates to refine the items in the workspace. For example, if the workspace is represented by a set of named clusters (name based upon major semantic content), the user may select a set of clusters that defines the next iteration of the search. An alternative use of aggregates is to correlate the search terms with items retrieved. Inspecting relevant and non-relevant items in a form that highlights the effect of the expanded search terms provides insights on what terms were the major causes for the results. A user may have thought a particular term was very important. A visual display could show that the term in fact had a minimal effect on the item selection process, suggesting a need to substitute other search terms.

Using a textual display on the results of a search provides no mechanism to display inter-relationships between items. For example, if the user is interested in the development of a polio vaccine, there is no way for a textual listing of found items to show “date” and “researcher” relationships based upon published items. The textual summary list of the Hit file can only be sorted via one attribute, typically relevance rank.

15.2 COGNITION AND PERCEPTION

Human cognition: Aspects of human cognition are the technical basis for understanding the details of information visualization systems. Many techniques are being developed heuristically with the correlation to human cognition and perception analyzed after the techniques are in test. The commercial pressures to provide visualization in delivered systems places the creativity under the intuitive concepts of the developer. The user-machine interface has primarily focused on a paradigm of a typewriter. As computers displays became ubiquitous, man-machine interfaces focused on treating the display as an extension of paper

with the focus on consistency of operations. The evolution of the interface focused on how to represent to the user what is taking place in the computer environment. The advancements in computer technology, information sciences and understanding human information processing are providing the basis for extending the human computer interface to improve the information flow, thus reducing wasted user overhead in locating needed information.

Although the major focus is on enhanced visualization of information, other senses are also being looked at for future interfaces. The audio sense has always been part of simple alerts in computers. Illegal inputs are usually associated with a beep, and more recently users have a spectrum of audio sounds to associate with everything from start-up to shut down. The sounds are now being replaced by speech in both input and output interfaces. The tactile (touch) sense is being addressed in the experiments using Virtual Reality. For example, VR is used as a training environment for areas such as medical procedures where tactile feedback plays an increasing role.

Background: A significant portion of the brain is devoted to vision and supports the maximum information transfer function from the environment to a human being. The center of debates in the 1970s was whether vision should be considered data collection or also has aspects of information processing.

In 1969, Arnheim questioned the then current psychological division of cognitive operations of perception and thinking as separate processes [Arnheim69]. Until then perception was considered a data collection task and thinking as a higher level function using the data. He contended that visual perception includes the process of understanding the information, providing an ongoing feedback mechanism between the perception and thinking. He further expanded his views arguing that treating perception and thinking as separate functions treats the mind as serial automata [Arnheim86]. Under this paradigm, the two mental functions exclude each other, with perception dealing with individual instances versus generalizations. Visualization is the transformation of information into a visual form which enables the user to observe and understand the information. This concept can be extended where the visual images provide a fundamentally different way to understand information that treats the visual input not as discrete facts but as an understanding process. The Gestalt psychologists postulate that the mind follows a set of rules to combine the input stimuli to a mental representation that differs from the sum of the individual inputs [Rock90]:

- Proximity - nearby figures are grouped together

- Similarity - similar figures are grouped together
- Continuity - figures are interpreted as smooth continuous patterns rather than discontinuous concatenations of shapes (e.g., a circle with its diameter drawn is perceived as two continuous shapes, a circle and a line, versus two half circles concatenated together)
- Closure - gaps within a figure are filled in to create a whole (e.g., using dashed lines to represent a square does not prevent understanding it as a square)
- Connectedness - uniform and linked spots, lines or areas are perceived as a single unit

Shifting the information processing load from slower cognitive processes to faster perceptual systems significantly improves the information-carrying interfaces between humans and computers. There are many ways to present information in the visual space. An understanding of the way the cognitive processes work provides insights for the decisions on which of the presentations will maximize the information passing and understanding. There is not a single correct answer on the best way to present information.

Aspects of the Visualization Process: The different criteria's that affect the visualization process are explained below.

Pre-attention towards primitives: One of the first-level cognitive processes is pre-attention, that is, taking the significant visual information from the photoreceptors and forming primitives. Primitives are part of the preconscious processes that consist of involuntary lower order information processing. An example of this is the ease with which our visual systems detect borders between changes in orientation of the same object. In Figure 8.1 the visual system quickly detects the difference in orientations between the left and middle portion of the figure and determines the logical border between them. This is not true for right and middle portion of the Figure 15.1.

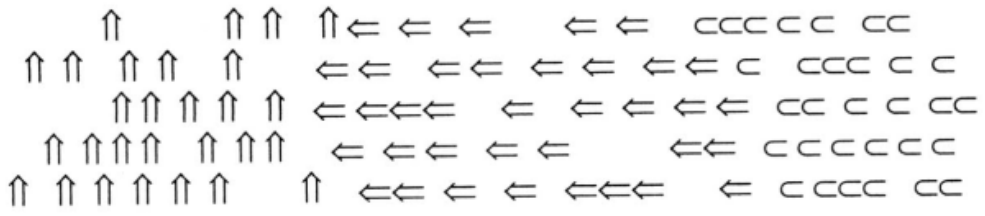


Figure 15.1 Pre-attentive Detection Mechanisms

This suggests that if information semantics are placed in orientations, the mind's clustering aggregate function enables detection of groupings easier than using different objects (assuming the orientations are significant). This approach makes maximum use of the feature detectors in the retina.

The pre-attentive process can detect the boundaries between orientation groups of the same object. A harder process is to identify the equivalence of rotated objects. For example, a rotated square requires more effort to recognize it as a square. As we migrate into characters, the problem of identification of the character is affected by rotating the character in a direction not normally encountered. It is easier to detect the symmetry when the axis is vertical. Figure 15.2 demonstrates these effects.

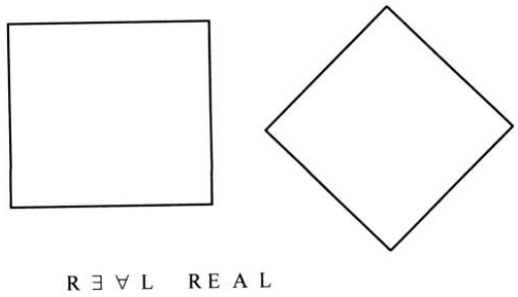


Figure 15.2 Rotating a Square and Reversing Letters in "REAL"

Optical illusion due to backround: Another visual factor is the optical illusion that makes a light object on a dark background to appear larger than if the item is dark and the background is light. Making use of this factor suggests that a visual display of small objects should use bright colors. An even more complex area is the use of colors. Colors have many attributes that can be modified such as hue, saturation and lightness. Hue is the physiological attribute of color sensation. Saturation is the degree to which a hue is different from a gray line with the same lightness, while lightness is the sensation of the amount of white or black. Complementary colors are two colors that form white or gray when combined (red/green, yellow/blue). Color is one of the most frequently used visualization techniques to organize, classify, and enhance features. Humans have an innate attraction to the primary colors (red, blue, green and yellow), and their retention of images associated with these colors is longer. But colors also affect emotion, and some people have strong aversion to certain colors. The negative side of use of colors is that some people are color blind to some or many colors. Thus any display that uses colors should have other options available.

Depth: Depth, like color, is frequently used for representing visual information. Classified as monocular cues, changes in shading, blurring (proportional to distance), perspective, motion, stereoscopic vision, occlusion and texture depict depth. Most of the cues are affected more by lightness than contrast. Thus, choice of colors that maximizes brightness in contrast to the background can assist in presenting depth as a mechanism for representing information. Depth has the advantage that depth/size recognition are learned early in life and used all of the time. The visual information processing system is attuned to processing information using depth and correlating it to real world paradigms.

Configural aspects of a display [Rose-95]. A configural effect occurs when arrangements of objects are presented to the user allowing for easy recognition of a high-level abstract condition. Configural clues substitute a lower level visual process for a higher level one that requires more concentration. These clues are frequently used to detect changes from a normal operating environment such as in monitoring an operational system. An example is shown in Figure 15.3 where the sides of a regular polygon (e.g., a square in this example) are modified. The visual processing system quickly detects deviations from normally equally sized objects.

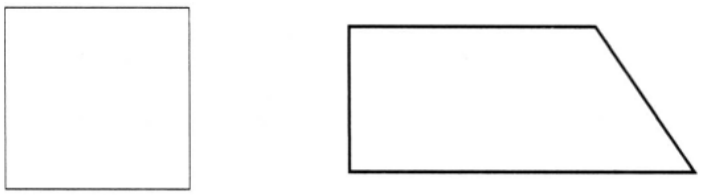


Figure 15.3 Distortions of a Regular Polygon

Spatial frequency: The human visual and cognitive system tends towards order and builds an coherent visual image whenever possible. The multiple spatial channel theory proposes that a complex image is constructed from the external inputs, not received as a single image. The final image is constructed from multiple receptors that detect changes in spatial frequency, orientation, contrast, and spatial phase. Spatial frequency is an acuity measure relative to regular light-dark changes that are in the visual field or similar channels. A cycle is one complete light-dark change. The spatial frequency is the number of cycles per one degree of visual field. Our visual systems are less sensitive to spatial frequencies of about 5-6 cycles per degree of visual field. One degree of visual field is approximately the viewing angle subtended by the width of a finger at arm's length. Other animals have significantly more sensitive systems that allow them to detect outlines of camouflaged prey not detected by humans until we focus on the area. Associated with not processing the higher spatial

frequencies is a reduction in the cognitive processing time, allowing animals (e.g. cats) to react faster to motion. When looking at a distinct, well defined image versus a blurred image, our visual system will detect motion/changes in the distinct image easier than the blurred image. If motion is being used as a way of aggregating and displaying information, certain spatial frequencies facilitate extraction of patterns of interest.

Learning from usage: The human sensory systems learn from usage. In deciding upon visual information techniques, parallels need to be made between what is being used to represent information and encountering those techniques in the real world environment. The human system is adept at working with horizontal and vertical references. They are easily detected and processed. Using other orientations requires additional cognitive processes to understand the changes from the expected inputs. The typical color environment is subdued without large areas of bright colors. Thus using an analogous situation, bright colors represent items to be focused on correlating to normal processing (i.e., noticing brightly colored flowers in a garden). Another example of taking advantage of sensory information that the brain is use to processing is terrain and depth information. Using a graphical representation that uses depth of rectangular objects to represent information is an image that the visual system is used to processing. Movement in that space is more easily interpreted and understood by the cognitive processes than if, for example, a three-dimensional image of a sphere represented a visual information space.

User's background and context of the information: In using cognitive engineering in designing information visualization techniques, a hidden risk is that "understanding is in the eye of the beholder." The integration of the visual cues into an interpretation of what is being seen is also based upon the user's background and context of the information. The human mind uses the latest information to assist in interpreting new information. If a particular shape has been representing important information, the mind has a predisposition to interpret new inputs as the same shape. For example, if users have been focusing on clusters of items, they may see clusters in a new presentation that do not exist. This leads to the question of changing visualization presentations to minimize legacy dispositions. Another issue is that our past experiences can affect our interpretation of a graphic. Users may interpret figures according to what is most common in their life experiences rather than what the designer intended.

15.3 INFORMATION VISUALIZATION TECHNOLOGIES

Introduction: The theories associated with information visualization are being applied in commercial and experimental systems to determine the best way to improve the user interface, facilitating the localization of information. They have been applied to many different situations and environments (e.g., weather forecasting to architectural design). The ones focused on Information Retrieval Systems are investigating how best to display the results of searches, structured data from DBMSs and the results of link analysis correlating data.

The goals for displaying the result from searches fall into two major classes: document *clustering* and *search statement analysis*.

- The goal of document clustering is to present the user with a visual representation of the document space constrained by the search criteria. Within this constrained space there exist clusters of documents defined by the document content. Visualization tools in this area attempt to display the clusters, with an indication of their size and topic, as a basis for users to navigate to items of interest. This is equivalent to searching the index at a library and then pursuing all the books on the different shelf locations that are retrieved by the search.
- The second goal is to assist the user in understanding why items were retrieved, thereby providing information needed to refine the query. Unlike the traditional Boolean systems where the user can easily correlate the query to the retrieved set of items, modern search algorithms and their associated ranking techniques make it difficult to understand the impacts of the expanded words in the search statement. Visualization techniques approach this problem by displaying the total set of terms, including additional terms from relevance feedback or thesaurus expansions, along with documents retrieved and indicate the importance of the term to the retrieval and ranking process. Structured databases are important

Structured databases are important to information retrieval because structured tiles are the best implementation to hold certain citation and semantic data that describe documents. Link analysis is also important because it provides aggregate-level information within an information system. Rather than treating each item as independent, link analysis considers information flowing between documents with value in the correlation between multiple

documents. For example, a time/event link analysis correlates multiple documents discussing a oil spill caused by a tanker. Even if all of the items retrieved on the topic are relevant, displaying the documents correlated by time may show dependencies of events that are of information importance and are not described in any specific document.

The goal of many visualization techniques is to show the semantic relationships between individual items to assist the user in locating those groups of items of interest. Another objective of visualization is in assisting the users in refining their search statements. It is difficult for users in systems using similarity measures to determine the primary causes for the selection and ranking of items in a Hit file. The automatic expansion of terms and intricacies of the similarity algorithms can make it difficult to determine the effects that the various words in the search statement are having on creating the Hit file. Visualization tools need to assist the user in understanding the effects of his search statement even to the level of identifying important terms that are not contributing to the search process. One solution is a graphical display of the characteristics of the retrieved items which contributed to their selection.

Visualization Technologies: In the remaining section we summarize some of the major techniques assisting for visual perception.

1D technique: 1D technique visualizes data, possibly of high dimensionality, through their linear view. Facet Map is one such 1D technique. Facet maps provides efficient searching among data items when search target is not exactly clear but can be identified by refining initial wide search query.

2D techniques: *Scatter plot:* Two-dimensional data can be visualized in different ways. A very common visualization form is the scatter plot. In a scatter plot the frame for the data presentation is a Cartesian coordinate system, in which the axes correspond to the two dimensions. The data is usually represented by points in the coordinate system's first quadrant (assuming the data point values are not negative). In case of two or more data sets being displayed in the same coordinate system different colours can be used to distinguish between the distinct plots. A problem with this way of displaying data arises when the amount of data points gets very high as the points become too dense. In order to avoid this Becker suggests binning of the data set [Sahling03]. The quality of the visualization now depends on the number of bins and their sizes. Figure 8.4 shows the distribution of miles per gallon vs. horsepower for American (red), European (blue) and Japanese (green) cars.

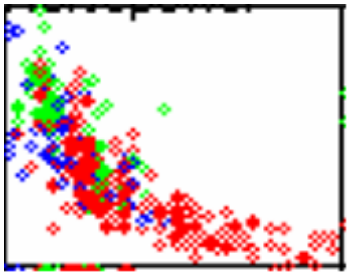


Figure 15.4: Scatter plot of car data set

Line graph: Another important visualization technique for two-dimensional data is the line graph. The difference to scatter plots is that this time the relation between the dimension on the horizontal axis and the one on the vertical axis is definite. The figure 8.5 shows an example for a line graph displaying the number of crimes in Niedersachsen in the years 1993 to 2002.

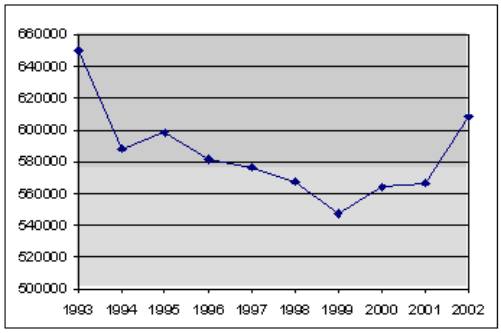


Figure 15.5: Total crimes (1993 - 2002)

Survey Plots: Extensions of line graphs are survey plots. They can be obtained by turning the plot 90 degrees clockwise and then halve the length of the rays and add this half on the other side of the now vertical axis. One more technique is the visualization of data as bar charts. Considering the last figure a bar chart representation would be the same as above but with the area under the graph filled in. Histograms are particular bar charts with the bar standing for the sum of the data point class.

3D techniques:

The two-dimensional techniques can easily be extended to three dimensions. The third dimension is achieved in scatter plots and bar charts by adding a further axis, orthogonal to the other two. The additional dimension in a line graph representation has the effect that the resulting plot is a surface. Figure 8.6 shows an example that has been generated with Matlab.

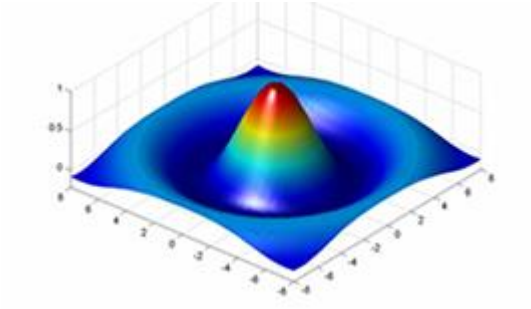


Figure 15.6: 3D line graph (surface)

High-dimensional data

The visualization of high-dimensional data raises a very severe problem: the visualization space is limited to three dimensions or even to only two since data is usually displayed on screens or paper. One of the obstacles in the discovery of high-dimensional data sets information is that techniques of extracting low-dimensional information and displaying it cannot automatically be employed for high-dimensional data as the data set size is too large.

Coming now to the different high-dimensional visualization techniques, we distinguish between icon-based, hierarchical and geometrical methods.

Icon-based methods: Icon-based methods are approaches that use icons (or glyphs) to represent high-dimensional data. They map data components to graphical attributes. The most famous technique is the use of Chernoff faces [Hoffmann02]. In this case a data point is represented by an individual face whereas the features map the data dimensions. Five different sizes of the eyes could correspond to the five products of the example above and the mouth might symbolize the two methods of payment. This scheme uses a person's ability of recognizing faces. Examples for Chernoff faces shows figure 15.7.

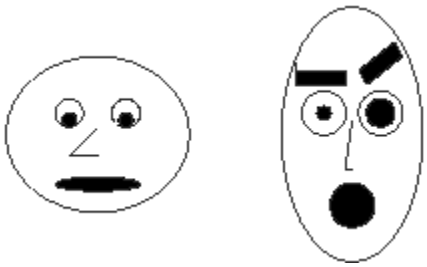


Figure 15.7: Chernoff faces

The probably most common icon-based technique is the use of **star glyphs** to denote data points. A star glyph consists of a centre point with equally angled rays. These branches correspond to the different dimensions and the length of the limbs mark the value of this

particular dimension for the studied data point. A polygon line connects the outer ends of the spokes [Oellien03]. An illustration of the star glyphs approach is figure 15.8.

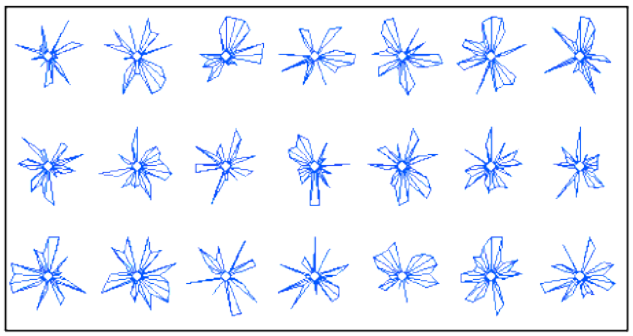


Figure 15.8: Star glyphs

These icon-based techniques are very vivid but have several disadvantages. A very severe problem is the organisation of the glyphs on the screen as no coordinate system representing two of the dimensions is provided. Even if you decided to use a Cartesian system it would put more weight on these two dimensions and so probably distort the data pattern. Another obstacle is the amount of variables and the size of the data set itself. If the number of rays become too high a distinction between the different spokes and the values they represent is not possible anymore. A similar unclear map emerges if the number of data points exceeds a certain amount.

Hierarchical representation: One way of organizing information is hierarchical. A tree structure is useful in representing information that ranges over time (e.g., genealogical lineage), constituents of a larger unit (e.g., organization structures, mechanical device definitions) and aggregates from the higher to lower level (e.g., hierarchical clustering of documents). A two-dimensional representation becomes difficult for a user to understand as the hierarchy becomes large.

The *Cone-Tree* is a 3-Dimensional representation of data, where one node of the tree is represented at the apex and ail the information subordinate to it is arranged in a circular structure at its base. Any child node may also be the parent of another cone. Selecting a particular node, rotates it to the front of the display. Compared to other hierarchical representations (e.g., node and link trees) the cone makes the maximum information available to the user providing a perspective on size of each of the subtrees.

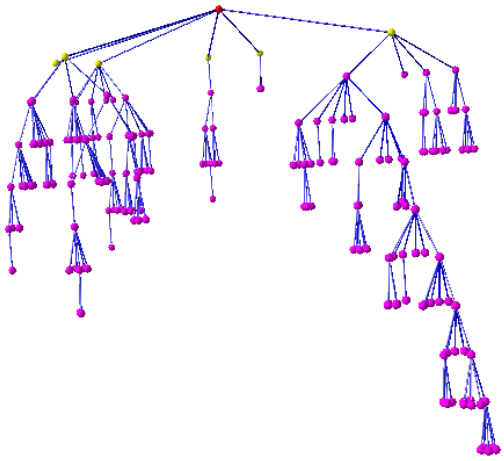


Figure 15.9: The 3D cone tree.

The most important representative of the group of hierarchical visualization techniques is dimensional stacking. Consider again the example with the five products, five territories, two sales channels, two methods of payment and five quarters [Mihalisin02]. First of all you have to select the two outermost dimensions. We choose the quarters and the pay types. Our horizontal axis is now divided into five parts while the vertical axis becomes halved. We now decide that we would like the sales channel to be embedded into the method of payment, so each part of the pay type axis gets further divided into two parts that represent the different channels. The axis corresponding to the quarters will embed the products so these elements become subdivided as well. Finally the upright axis lodges the five territories. The resulting coordinate axes combination system is the following.

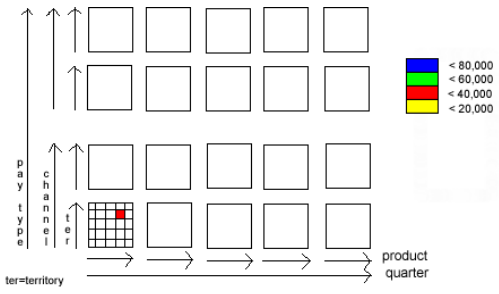


Figure 15.10: Dimensional stacking

Geometrical methods

Geometrical methods are a very large group of visualization techniques. Probably the easiest and most commonly used one is the method of **parallel coordinates**. Here the dimensions are represented by parallel lines, which are equally spaced. They are linearly scaled so that the bottom of the axis stands for the lowest possible value whereas the top corresponds to the highest value. A data point is now drawn into this system of axes with a polygonal line,

which crosses the variable lines at the locations the data point holds for the examined dimension. A simple example with three points and four dimensions is shown below: The points displayed are A = (1; 3; 2; 5), B = (2; 4; 1; 6) and C = (1; 4; 3; 5)

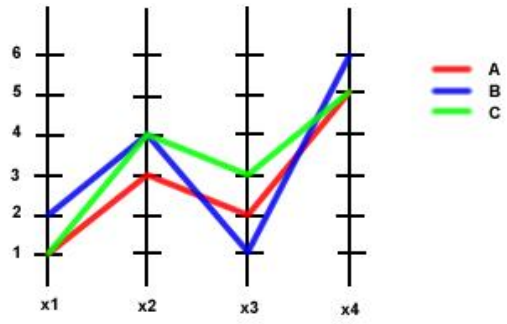


Figure 15.11: Parallel coordinates for data points A, B, C

Radial Coordinate Visualization uses the spring paradigm [Hoffmann02]. From a centre point n equally spaced limbs of the same length spread out, each representing one dimension. The ends of the lines mark the dimensional anchor of the respective variable, which are connected forming a circle. Before the data points can be visualized by this technique they need to be normalized. After that one end of a spring is fastened to each dimensional anchor, the other end to the data point. The spring constant of each spring is the value of the data point of the respective dimension. In order to determine the location of the data point the sum of the spring forces needs to equal zero. If you apply this method to the well known Iris data set you can obtain figure 15.12.

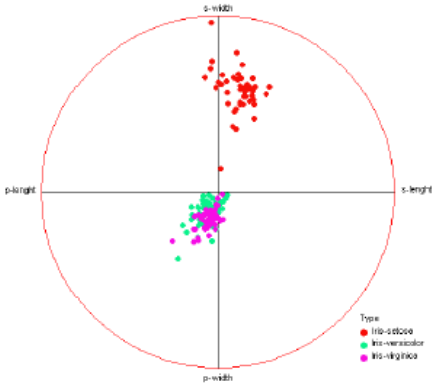


Figure 15.12: RadViz (Iris data set)

An advantage of RadViz is the fact that it preserves certain symmetries of the data set. The major disadvantage is the overlap of points.

15.4 SUMMARY

Information visualization is not a new concept. The well known saying that “a picture is worth a thousand words” is part of our daily life. Everything from advertisements to briefings makes use of visual aides to significantly increase the amount of information presented and provide maximum impact on the audience. The significant amount of “noise” (non-relevant items) in interactions with information systems requires use of user interface aides to maximize the information being presented to the user. Pure textual interfaces provide no capabilities for aggregation of data, allowing a user to see an overview of the results of a search. Viewing the results of a search using a hierarchical paradigm allows higher levels of abstraction showing overall results of searches before the details consume the display.

Visualization techniques attempt to represent aggregate information using a metaphor (e.g., peaks, valleys, cityscapes) to highlight the major concepts of the aggregation and relationships between them. This allows the user to put into perspective the total information before pursuing the details. It also allows major pruning of areas of non-relevant information. A close analogy is when searching on “fields” in the index at a library, the book shelves on horticulture would be ignored if magnetic fields were the information need. By having the data visualized constrained by the users search, the display is focused on the user’s areas of interest. Relationships between data and effectiveness of the search become obvious to the user before the details of individual items hide the higher level relationships.

Cognitive engineering suggests that alternative representations are needed to take maximum advantage of different physiological and cultural experiences of the user. Colors are useless to a color blind user. Using visual cues that a person has developed over his life-experience can facilitate the mapping of the visual metaphor to the information it is representing.

As the algorithms and automatic search expansion techniques become more complex, use of visualization will take on additional responsibilities in clarifying to the user, not only what information is being retrieved, but the relationship between the search statement and the items. Showing relationships between items has had limited use in systems and been focused on data mining type efforts. The growth of hypertext linkages will require new visualization tools to present the network relationships between linked items and assist the user in navigating this new structure.

15.5 QUESTIONS

1. Describe the need for information visualization.
2. Discuss the limits associated with use of pre-attentive processes, configural aspects, and spatial frequency as a basis for information visualization.
3. Describe what cognitive engineering principles are being used in the visualization techniques.
4. Briefly explain 1D and 2D visualization techniques.
5. Write a note on 3D visualization techniques.
6. Explain visualization using hierarchical representation.

15.6 KEYWORDS

Information Visualization, Cognitive engineering, Cognition and Perception, Cone Tree, Scatterplot.

15.7 REFERENCES FOR FURTHER READING/STUDIES

[Amheim69] Arnheim, R., "Visual Thinking", University of California Press, 1969

[Amheim86] Arnheim, R., "New Essays on the Psychology of Art", California Press, 1986.

[Mihalisin02] Mihalisin, T. Data Warfare and Multidimensional Education. In: Information Visualization in Data Mining and Knowledge Discovery. Morgan Kaufmann, San Francisco (CA) 2002

[Rock90] Rock, I and S. Palmer, "The Legacy of Gestalt Psychology", Scientific American, December, 1990, pages 84-90.

[Rose95] Rose, R. ed., "P1000 Science and Technology Strategy for Information Visualization", Version 1.6, August 1995.

[Rose96] Rose, R. ed., "P1000 Science and Technology Strategy for Information Visualization", Version 2, 1996.

UNIT 16: INFORMATION SYSTEM EVALUATION

Structure

16.1 INTRODUCTION TO INFORMATION SYSTEM EVALUATION

The importance of developing a robust and responsive information technology (IT) and information system (IS) infrastructure to support the formal planning and control of business processes is increasing. The necessity to evaluate the functionality performances of Information System has emerged from the importance of Information Technology in effectiveness and efficiency of work processes in an organization, causing rapid growth of demands in terms of resources performances in Information System.

Evaluation of Information System performances means evaluation of performances in hardware, software, computer networks, data and human resources. The main purpose of Information System functionality performances evaluation is upgrading and especially improvement in quality of maintenance.

The Information System functionality evaluation represents the procedure of assessing how successfully Information System fulfills its objectives. The process of evaluation includes synthesizing and determining gathered individual scores with the purpose of forming common opinion about the functionality of evaluated Information System. In the process of expressing general opinion professionals usually rely on their individual assessment abilities. Quality assessment and Information System oversights are done with the purpose of organization's Information System resources preservation and data integrity.

The most important factors that influence the success of an information system are:

- Functionality of Information System
- Data quality
- Expected usefulness of Information System
- Expected usage simplicity of Information System
- Self-efficiency of Information System user
- Usage of Information System
- Influence of information system on individuals
- Information System user's satisfaction
- Organizational factors

In recent years the evaluation of Information Retrieval Systems and techniques for indexing, sorting, searching and retrieving information have become increasingly important. This growth in interest is due to two major reasons: the growing number of retrieval systems being used and additional focus on evaluation methods themselves.

The Internet is an example of an information space (infospace) whose text content is growing exponentially along with products to find information for value. Information retrieval technologies are the basis behind the search of information on the Internet. In parallel with the commercial interest, the introduction of a large standardized test database and a forum for yearly analysis via TREC has provided a methodology for evaluating the performance of algorithms and systems.

From an academic perspective, measurements are focused on the specific effectiveness of a system and usually are applied for determining the effects of changing a system's algorithms or comparing algorithms among systems. From a commercial perspective, measurements are also focused on availability and reliability. For academic purposes, controlled environments can be created that minimize errors in data. In operational systems, there is no control over the users and care must be taken to ensure the data collected are meaningful.

Some of the reasons for evaluating the effectiveness of an Information Retrieval System are:

- To aid in the selection of a system to procure
- To monitor and evaluate system effectiveness

- To evaluate query generation process for improvements
- To provide inputs to cost-benefit analysis of an information system
- To determine the effects of changes made to an existing information system.

The most important evaluation metrics of information systems will always be biased by human subjectivity. This problem arises from the specific data collected to measure the user resources in locating relevant information. Metrics to accurately measure user resources expended in information retrieval are inherently inaccurate. The metrics used in determining how well a system is working is the **relevancy of items**. Relevancy of an item, however, is not a binary evaluation, but a continuous function between an item's being exactly what is being looked for and it's being totally unrelated.

From a human judgment standpoint, relevancy can be considered:

- **Subjective** depends upon a specific user's judgment
- **Situational** relates to a user's requirements
- **Cognitive** depends on human perception and behavior
- **Temporal** changes over time
- **Measurable** observable at points in time

In a dynamic environment, each user has his own understanding of the requirement and the threshold on what is acceptable. Based upon his cognitive model of the information space and the problem, the user judges a particular item. Some users consider information they already known to be non-relevant to their information need. Relevance judgment is measurable at a point in time constrained by the particular users and their thresholds on acceptability of information.

Another way of specifying relevance is from information, system and situational views.

- The ***information view*** is subjective in nature and pertains to human judgment of the conceptual relatedness between an item and the search. It involves the user's personal judgment of the relevancy (**aboutness**) of the item to the user's information need. When the reference experts (librarians, researchers, subject specialists, indexers)

assist the user, it is assumed they can reasonably predict whether certain information will satisfy the user's needs.

- The ***system view*** relates to a match between query terms and terms within an item. It can be objectively observed, manipulated and tested without relying on human judgment because it uses metrics associated with the matching of the query to the item.
- The ***situation view*** pertains to the relationship between information and the user's information problem situation. It assumes that only users can make valid judgments regarding the suitability of information to solve their information need.

Ingwersen categorizes the information view into four types of "aboutness":

- **Author Aboutness** Determined by the author's language as matched by the system in natural language retrieval.
- **Indexer Aboutness** Determined by the indexer's transformation of the author's natural language into a controlled vocabulary.
- **Request Aboutness** Determined by the user's or intermediary's processing of a search statement into a query.
- **User Aboutness** Determined by the indexer's attempt to represent the document according to presupposition about what the user will want to know.

Pertinence can be defined as those items that satisfy the user's information need at the time of retrieval. The TREC evaluation process uses relevance versus pertinence as its criteria for judging items because pertinence is too variable to attempt to measure in meaningful items (i.e., it depends on each situation).

16.2 MEASURES USED IN SYSTEM EVALUATIONS

To measure ad hoc information retrieval effectiveness in the standard way, we need a test collection consisting of three things:

1. A document collection
2. A test suite of information needs, expressible as queries

3. A set of relevance judgments, a binary assessment of
   either *relevant* or *nonrelevant* for each query-document pair.

The standard approach to information retrieval system evaluation revolves around the notion of *relevant* and *nonrelevant* documents. With respect to a user information need, a document in the test collection is given a binary classification as either relevant or nonrelevant. This decision is referred to as the *gold standard* or *ground truth* judgment of relevance. The test document collection and suite of information needs should be reasonable size.

To define the measures that can be used in evaluating Information Retrieval Systems, it is useful to define the major functions associated with identifying relevant items in an information system.

For example, an information need might be:

Information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.

This might be translated into a query such as:

wine and red and white and heart and attack and effective

A document is relevant if it addresses the stated information need, not because it just happens to contain all the words in the query. But, nevertheless, an information need is present.

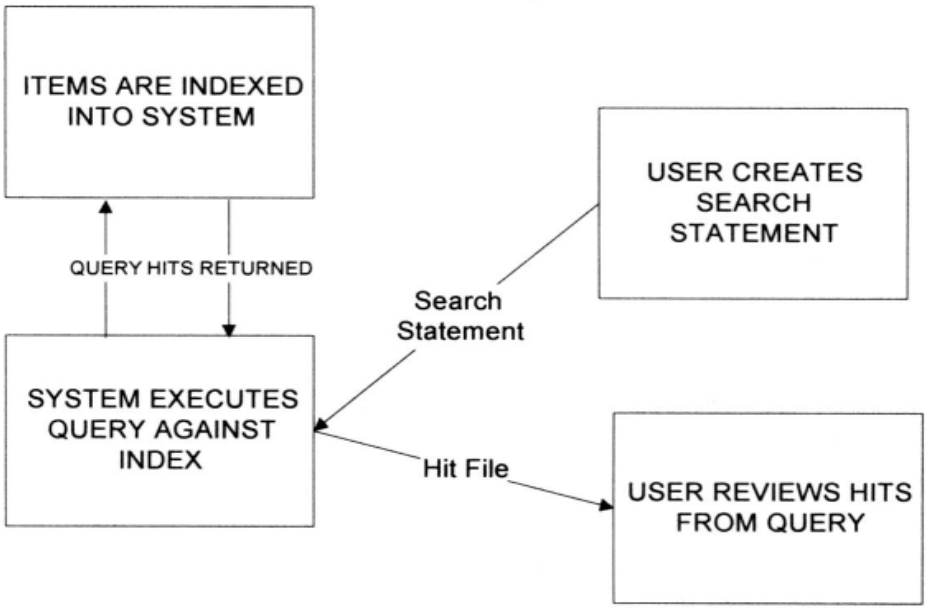


Figure 16.1 Identifying Relevant Items

Items that arrive in the system are automatically or manually transformed by "indexing" into searchable data structures. The user determines what the information need is and creates a search statement. The system processes the search statement, returning potential hits. The user selects those hits to review and accesses them.

Text retrieval engines, commonly known as search engines with examples such as google.com and google.com, return a list of documents (the **hitlist***)* for a query. Typically, there are some good documents (the ones users wanted) in the list and some bad ones. The quality of a search engine is measured in terms of the proportion of good hits in the list, the positions of good hits relative to bad ones, and the proportion of good documents missing from the list.

Measurements can be made from two perspectives: user perspective and system perspective. The **Author's Aboutness** occurs as part of the system executing the query against the index. The **Indexer Aboutness** and **User Aboutness** occur when the items are indexed into the system. The **Request Aboutness** occurs when the user creates the search statement. The ambiguities in the definition of what is relevant occur when the user is reviewing the hits from the query.

Techniques for collecting measurements can also be **objective or subjective**. An objective measure is one that is well-defined and based upon numeric values derived from the system operation. A subjective measure can produce a number, but is based upon an individual user judgment.

Measurements with automatic indexing of items arriving at a system are derived from **standard performance monitoring** associated with any program in a computer (e.g., resources used such as memory and processing cycles) and time to process an item from arrival to availability to a search process. When manual indexing is required, the measures are then associated with the **indexing process**. The measure is usually defined in terms of time to index an item. The value is normalized by the exhaustivity and specificity requirements.

Another measure in both the automatic and manual indexing process is the **completeness** and **accuracy of the indexes created**. These are evaluated by random sampling of indexes by

quality assurance personnel. A more complex area of measurements is associated with the search process. This is associated with a user creating a new search or modifying an existing query. In creating a search, an example of an objective measure is the time required to create the query, measured from when the user enters into a function allowing query input to when the query is complete. Completeness is defined when the query is executed.

Response time is a metric frequently collected to determine the efficiency of the search execution. Response time is defined as the time it takes to execute the search. The beginning is always correlated to when the user tells the system to begin searching. The end time is affected by the difference between the user's view and a system view. From a user's perspective, a search could be considered complete when the first result is available for the user to review, especially if the system has new items available whenever a user needs to see the next item. From a system perspective, system resources are being used until the search has determined all hits.

To ensure **consistency**, response time is usually associated with the completion of the search. This is one of the most important measurements in a production system. Determining how well a system is working answers the typical concern of a user: "the system is working slow today."

Data are usually gathered on the search creation and Hit file review process by **subjective techniques**, such as ***questionnaires*** to evaluate system effectiveness.

Correctness, Relevance and Effectiveness

Relevance is somewhat subjective and similar to any interpretation of a natural language text. Relevance judgments of hit lists made by users can be used to compare different retrieval engines. They can also be used to measure the quality of a particular engine within the limits of statistical validation techniques.

Effectiveness of a system is measured by its ability to satisfy the user. Traditionally, these are measured in terms of ***precision and recall, fallout and generality.*** The measures require a collection of documents and a query. All these measures assume a ground truth notion of relevancy: every document is known to be either relevant or non-relevant to a particular query. In practice queries may be ill-posed and there may be different shades of relevancy.

Precision

Precision is a measure of the accuracy of the search process. It directly evaluates the correlation of the query to the database and indirectly is a measure of the completeness of the indexing algorithm. Precision is the fraction of the documents retrieved that are relevant to the user's information need.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

In binary classification, precision is analogous to positive predictive value. Precision takes all retrieved documents into account. It can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called *precision at n* or *P@n*.

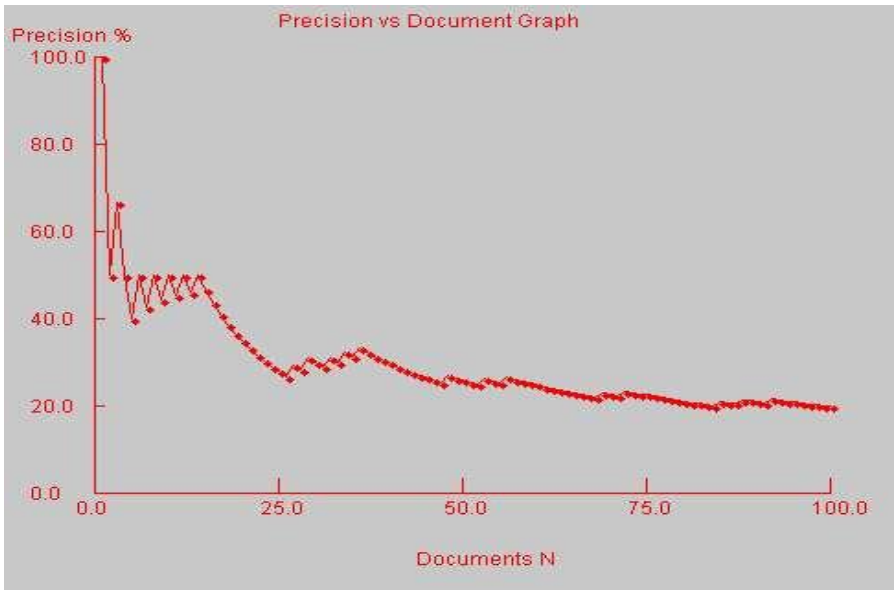


Figure 16.2: Precision versus Document graph

Precision is a measure of how well the engine performs in not returning nonrelevant documents. Precision is 100% when every document returned to the user is relevant to the query. There is no easy way to achieve 100% precision other than in the trivial case where no document is ever returned for any query!

Recall

Recall is a measure of the ability of the search to find all of the relevant items that are in the database. Recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

In binary classification, recall is often called sensitivity. So it can be looked at as *the probability that a relevant document is retrieved by the query*. It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision.

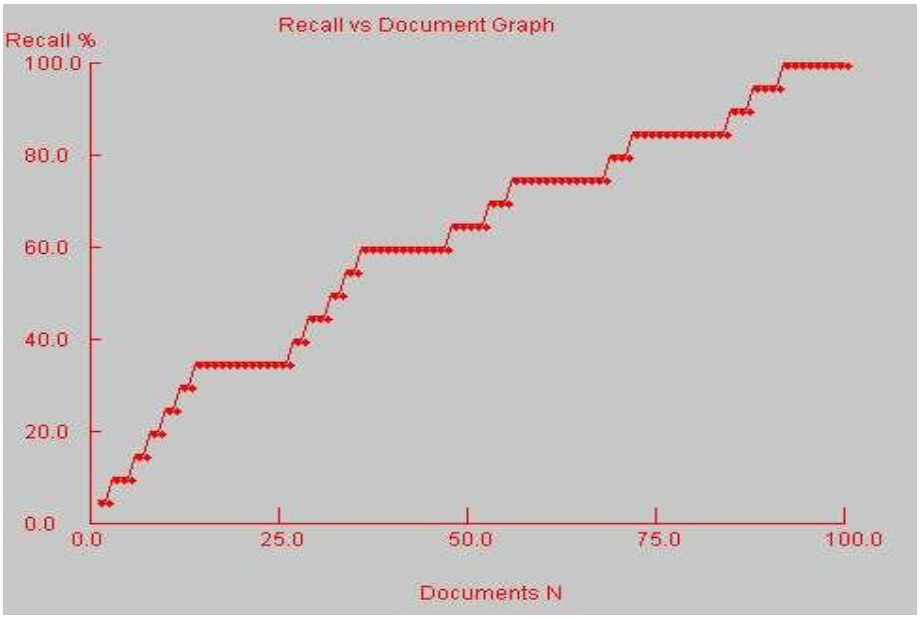


Figure 16.3: Recall versus Document graph

Recall is a measure of how well the engine performs in finding relevant documents. Recall is 100% when every relevant document is retrieved. In theory, it is easy to achieve good recall: simply return every document in the collection for every query. Therefore, recall by itself is not a good measure of the quality of a search engine.

For a given information system, the relationship between precision and recall could be described by the following graph.

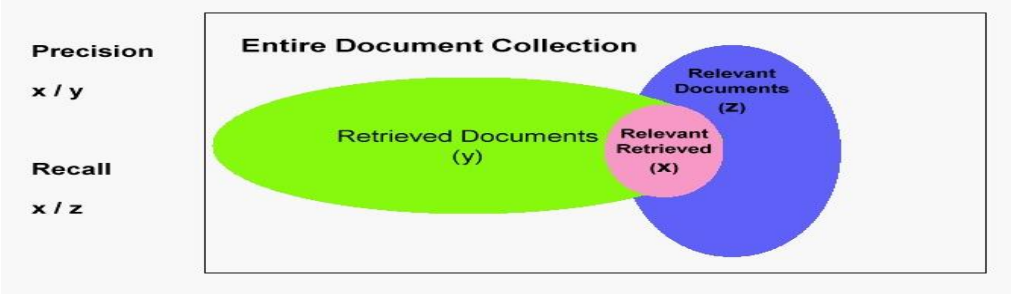


Figure 16.4: Relationship between precision and recall

A retrieval engine must have a high recall to be admissible in most applications. The key in building better retrieval engines is to increase precision without sacrificing recall. Most search engines on the Web, for example, give a reasonably good recall but poor precision, i.e., they find some relevant documents but they also return too many nonrelevant hits that users do not want to read.

Fall-out

Another measure that is directly related to retrieving non-relevant items can be used in defining how effective an information system is operating. This measure is called **Fallout.**

The proportion of non-relevant documents that are retrieved, out of all non-relevant documents available:

$$\text{fall-out} = \frac{|\{\text{non-relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{non-relevant documents}\}|}$$

In binary classification, fall-out is closely related to specificity and is equal to $(1 - \text{specificity})$. It can be looked at as *the probability that a non-relevant document is retrieved by the query*. It is trivial to achieve fall-out of 0% by returning zero documents in response to any query.

Fallout can be viewed as the inverse of recall and will never encounter the situation of 0/0 unless all the items in the database are relevant to the search. It can be viewed as the probability that a retrieved item is non-relevant. Recall can be viewed as the probability that a retrieved item is relevant. From a system perspective, the ideal system demonstrates maximum recall and minimum fallout.

Of the three measures (precision, recall and fallout), fallout is least sensitive to the accuracy of the search process. The large value for the denominator requires significant changes in the number of retrieved items to affect the current value.

Generality

The proportion of relevant documents within the entire collection is defined as generality given by G = n1/N where n1 represents the number of relevant documents and N indicates total number of documents.

F-measure

The weighted harmonic mean of precision and recall, the traditional F-measure or balanced F-score is:

$$F = \frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{(\mathrm{precision} + \mathrm{recall})}.$$

This is also known as the F_1 measure, because recall and precision are evenly weighted.

The general formula for non-negative real β is:

$$F_\beta = \frac{(1 + \beta^2) \cdot (\mathrm{precision} \cdot \mathrm{recall})}{(\beta^2 \cdot \mathrm{precision} + \mathrm{recall})}.$$

Two other commonly used F measures are the F_2 measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

The F-measure was derived by van Rijsbergen (1979) so that F_β "measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision". It is based on van Rijsbergen's effectiveness measure

$$E = 1 - \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}}.$$

Their relationship is

$$F_\beta = 1 - E \text{ where } \alpha = \frac{1}{1 + \beta^2}.$$

Average precision

Precision and recall are single-value metrics based on the whole list of documents returned by the system. For systems that return a ranked sequence of documents, it is desirable to also consider the order in which the returned documents are presented. By computing a precision

and recall at every position in the ranked sequence of documents, one can plot a precision-recall curve, plotting precision $p(r)$ as a function of recall r.

Average precision computes the average value of $p(r)$ over the interval from $r = 0$ to $r = 1$.

$$\text{AveP} = \int_0^1 p(r)dr.$$

This integral is in practice replaced with a finite sum over every position in the ranked sequence of documents:

$$\text{AveP} = \sum_{k=1}^{n} P(k)\Delta r(k)$$

where k is the rank in the sequence of retrieved documents, n is the number of retrieved documents, $P(k)$ is the precision at cut-off k in the list, and $\Delta r(k)$ is the change in recall from items $k-1$ to k.

This finite sum is equivalent to:

$$\text{AveP} = \frac{\sum_{k=1}^{n}(P(k) \times rel(k))}{\text{number of relevant documents}}$$

where

$rel(k)$ is an indicator function equaling 1 if the item at rank k is a relevant document, zero otherwise. Note that the average is over all relevant documents and the relevant documents not retrieved get a precision score of zero.

Three-Point Average, Eleven-Point Average and Normalized Recall

A ***three-point average precision*** is computed by averaging the precision of the retrieval system for a given query at three defined recall levels. Typically theses recall levels are 0.25, 0.50 and 0.75.

The ***eleven-point average precision*** is computed using the recall levels 0.0, 0.1, 0.2, …, 0.9 and 1.0. The more points are used in computing the average precision, the more work it requires and the more accurate the computed value. Thus the eleven-point average requires a little more effort to compute than the three-point average but provides a more accurate value.

The impact of "wiggles" in the curve is reduced by interpolating the $p(r)$ function. For example, the PASCAL Visual Object Classes challenge (a benchmark for computer vision object detection) computes average precision by averaging the precision over a set of evenly spaced recall levels {0, 0.1, 0.2, ... 1.0}:

$$\mathrm{AveP} = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1.0\}} p_{\mathrm{interp}}(r)$$

where $p_{\mathrm{interp}}(r)$ is an interpolated precision that takes the maximum precision over all recalls greater than r:

$$p_{\mathrm{interp}}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r})$$.

An alternative is to derive an analytical $p(r)$ function by assuming a particular parametric distribution for the underlying decision values. For example, a *binormal precision-recall curve* can be obtained by assuming decision values in both classes to follow a Gaussian distribution. Average precision is also sometimes referred to geometrically as the area under the precision-recall curve.

Normalized recall is a measure in which recall is normalized against all relevant documents. Suppose there are N documents in the collection and out of which n are relevant. These n documents are ranked as i_1, i_2,…, i_n. Then the formula for computing the normalized recall is:

$$\textit{Normalize recall} = \mathrm{SUM}_{j = 1..n} (i_j - j) / (n * (N - n))$$

Precision-Recall Graphs

A common way to depict the degradation of precision at *n* as one traverses the hitlist is to plot interpolated precision numbers against percentage recall. A percentage recall of say 50% is the position in the hitlist at which 50% of the relevant documents in the collection (i.e., 0.5 *

R) have been retrieved. It is a measure of the number of hits you have to read before you have seen a certain percentage of relevant documents.

Precision-recall graphs have a classical concave shape. The following figure shows a typical graph. The graph shows the trade-off between precision and recall. Trying to increase recall typically introduces more bad hits into the hitlist, thereby reducing precision (i.e., moving to the right along the curve). Trying to increase precision typically reduces recall by removing some good hits from the hitlist (i.e., moving left along the curve).

An ideal goal for a retrieval engine is to increase both precision and recall by making improvements to the engine, i.e., the entire curve must move up and out to the right so that both recall and precision are higher at every point along the curve. Average precision is indeed the area under the precision-recall graph. By moving the curve up and to the right, the area under the graph increases, thereby increasing the average precision of the engine.

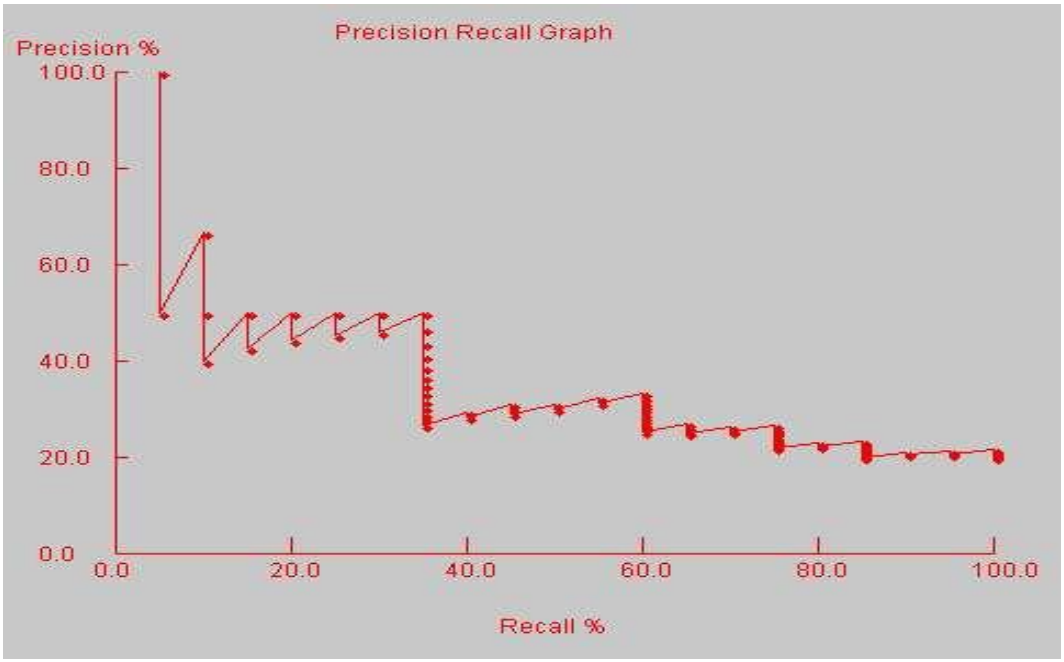


Figure 16.5 Precision Recall graph

R-Precision

Precision at **R**-th position in the ranking of results for a query that has **R** relevant documents. This measure is highly correlated to Average Precision. Also, Precision is equal to Recall at the **R**-th position.

Mean average precision

Mean average precision for a set of queries is the mean of the average precision scores for each query.

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{Q}$$

where *Q* is the number of queries.

Discounted cumulative gain

DCG uses a graded relevance scale of documents from the result set to evaluate the usefulness, or gain, of a document based on its position in the result list. The premise of DCG is that highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result.

The DCG accumulated at a particular rank position p is defined as:

$$\mathrm{DCG_p} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}.$$

Since result set may vary in size among different queries or systems, to compare performances the normalized version of DCG uses an ideal DCG. To this end, it sorts documents of a result list by relevance, producing an ideal DCG at position p ($IDCG_p$), which normalizes the score:

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCGp}.$$

The nDCG values for all queries can be averaged to obtain a measure of the average performance of a ranking algorithm. Note that in a perfect ranking algorithm, the DCG_p will be the same as the $IDCG_p$ producing an nDCG of 1.0. All nDCG calculations are then relative values on the interval 0.0 to 1.0 and so are cross-query comparable.

Two approaches have been suggested to gain the insights associated with testing a search against a large database.

- The first is to use a **sampling technique** across the database, performing relevance judgments on the returned items. This would form the basis for an estimate of the total relevant items in the database.

- The other technique is to **apply different search strategies to the same database for the same query**. An assumption is then made that all relevant items in the database will be found in the aggregate from all of the searches. This later technique is what is applied in the TREC-experiments.

Unique Relevance Recall

A new measure that provides additional insight in comparing systems or algorithms is the "Unique Relevance Recall" (URR) metric. URR is used to compare more two or more algorithms or systems. It measures the number of relevant items that are retrieved by one algorithm that are not retrieved by the others:

$$\text{Unique\_Relevance\_Recall} = \frac{Number\_unique\_relevant}{Number\_relevant}$$

Number_unique_relevant is the number of relevant items retrieved that were not retrieved by other algorithms. When many algorithms are being compared, the definition of *uniquely* found items for a particular system can be modified, allowing a small number of other systems to also find the same item and still be considered unique. This is accomplished by defining a percentage of the total number of systems that can find an item and still consider it unique. *Number_relevant* can take on two different values based upon the objective of the evaluation:

VALUE	INTERPRETATION
Total Number Retrieved Relevant (TNRR)	the total number of relevant items found by all algorithms
Total Unique Relevant Retrieved (TURR)	the total number of unique items found by all the algorithms

A	B	C	D	E	F	G	H	I	J	K	L	M
3	4	2	22	1	100	200	22	100	10	500	6	15

Figure 16.6a Number Relevant Items

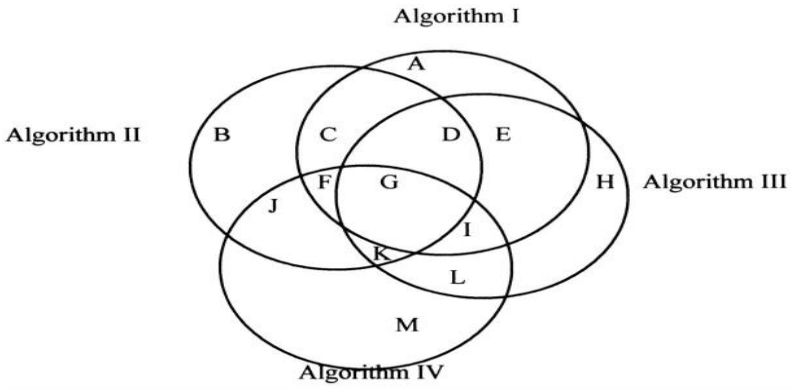


Figure 16.6b: Overlap of relevant items

Using TNRR as the denominator provides a measure for an algorithm of the percent of the total items that were found that are unique and found by that algorithm. It is a measure of the contribution of uniqueness to the total relevant items that the algorithm provides. Using the second measure, TURR, as the denominator, provides a measure of the percent of total unique items that could be found that are actually found by the algorithm.

Figure 16.6a and 16.6b provide an example of the overlap of relevant items assuming there are four different algorithms. Figure 16.6a gives the number of items in each area of the overlap diagram in Figure 16.6b. If a relevant item is found by only one or two techniques as a "unique item," then from the diagram the following values URR values can be produced:

Algorithm I - 6 unique items (areas A, C, E)
Algorithm II - 16 unique items (areas B, C, J)
Algorithm III - 29 unique items (areas E, H, L)
Algorithm IV - 31 unique items (areas J, L, M)

Algorithm I 6/985 = .0061 6/61= .098
Algorithm II 16/985 = .0162 16/61= .262
Algorithm I I I 29/985 = .0294 29/61 = .475
Algorithm IV 31/985 = .0315 31/61 = .508

The URR value is used in conjunction with Precision, Recall and Fallout to determine the total effectiveness of an algorithm compared to other algorithms. The URR TNRR value indicates what portion of all unique items retrieved by all of the algorithms was retrieved by a

specific algorithm. The URR TURR value indicates the portion of possible unique items that a particular algorithm found. In the example, Algorithm IV found 50 per cent of all unique items found across all the algorithms. The results indicate that if it is essential to increase the recall by running two algorithms, then choose algorithm III or IV in addition to the algorithm with the highest recall value. Like Precision, URR can be calculated since it is based upon the results of retrieval versus results based upon the complete database. It assists in determining the utility of using multiple search algorithms to improve overall system performance.

Other measures have been proposed for judging the results of searches:

- **Novelty Ratio** Ratio of relevant and not known to the user to total relevant retrieved.
- **Coverage Ratio** Ratio of relevant items retrieved to total relevant by the user before the search.
- **Sought Recall** Ratio of the total relevant reviewed by the user after the search to the total relevant the user would have liked to examine.

Utility measure

In some systems, programs filter text streams, software categorizes data or intelligent agents alert users if important items are found. In these systems, the Information Retrieval System makes decisions without any human input and their decisions are binary in nature (an item is acted upon or ignored). These systems are called binary classification systems for which effectiveness measurements are created to determine how algorithms are working. One measure is the utility measure that can be defined as:

$$U = \alpha*(\text{Relevant\_Retrieved}) + \beta*(\text{Non-Relevant\_Not Retrieved}) - \delta*(\text{Non-Relevant\_Retrieved}) - \gamma*(\text{Relevant\_Not Retrieved})$$

where and are positive weighting factors the user places on retrieving relevant items and not retrieving non-relevant items while and are factors associated with the negative weight of not retrieving relevant items or retrieving non-relevant items. This formula can be simplified to account only for retrieved items with and equal to zero.

E-measure

Another family of effectiveness measures called the **E-measure** that combines recall and precision into a single score was proposed by Van Rijsbergen.

The measures are optimal from a system perspective, and very useful in evaluating the effect of changes to search algorithms. The current evaluation metrics require a classification of items into relevant or non-relevant. When forced to make this decision, users have a different threshold. These leads to the suggestion that the existing evaluation formulas could benefit from extension to accommodate a spectrum of values for relevancy of an item versus a binary classification. But the innate issue of the subjective nature of relevant judgments will still exist, just at a different level. The Text REtrieval Conferences (TRECs), sponsored on a yearly basis, provide a source of a large "ground truth" database of documents, search statements and expected results from searches essential to evaluate algorithms. It also provides a yearly forum where developers of algorithms can share their techniques with their peers.

16.3 SUMMARY

Evaluation of Information Retrieval Systems is essential to understand the source of weaknesses in existing systems and tradeoffs between using different algorithms. The standard measures of Precision, Recall, and Fallout have been used for the last twenty-five years as the major measures of algorithmic effectiveness. With the insertion of information retrieval technologies into the commercial market and ever growing use on the Internet, other measures will be needed for real time monitoring the operations of systems.

16.4 KEYWORDS

Information system, Precision, Recall, Fallout, Generality, Relevance, Completeness

16.5 QUESTIONS

1. What are the problems associated with generalizing the results from controlled tests on information systems to their applicability to operational systems? Does this invalidate the utility of the controlled tests?
2. What are the main issues associated with the definition of relevance? How would you overcome these issues in a controlled test environment?

3. What techniques could be applied to evaluate each step in Figure 16.6?
4. Consider the following table of relevant items in ranked order from four algorithms along with the actual relevance of each item. Assume all algorithms have highest to lowest relevance is from left to right (Document 1 to last item). A value of zero implies the document was non-relevant).

Document	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Algo1	1	0	0	1	1	1	0	0	1	1	0	0	1	1
Algo2	0	1	1	0	1	1	1	0	0	1	1	0	1	1
Algo3	0	1	0	0	1	1	1	1	1	0	1	1	1	1
Actual	1	1	1	1	0	0	1	1	1	0	0	1	1	1

Document	15	16	17	18	19	20	21	22	23	24	25	26	27
Algo1	1	0	0	1	1	1	0	0	1	1	0	0	1
Algo2	0	1	1	0	1	1	1	0	0	1	1	0	1
Algo3	0	1	0	0	1	1	1	1	1	0	1	1	1
Actual	1	1	1	1	0	0	1	1	1	0	0	1	1

   a. Calculate and graph precision/recall for all the algorithms on one graph.
   b. Calculate and graph fallout/recall for all the algorithms on one graph
   c. Calculate the TNRR and TURR for each algorithm (assume uniquely found is only when one algorithm found a relevant item)
   d. Identify which algorithm is best and why.
5. What is the relationship between precision and TURR?

16.6 REFERENCES

Bird-78 – Bird, R., Newsbaum, J. and J. Trefftzs, "Text Files Inversion: An Evaluation", Proceedings of the Fourth Workshop on Computer Architecture for Non-Numeric Processing, Syracuse, N.Y., August 1-4, 1978, pages 42-50.

Croft-94 – Croft, W. B., Callan, J. and J. Broglio, "Trec-2 Routing and Ad Hoc Retrieval Evaluation Using the INQUERY System", in The Second Text Retrieval Conference (TREC-2) Proceedings, NIST publications, 1993.

Harper-78 – Harper, D. J. and C. J. van Rijsbergen, "An Evaluation of Feedback in Document Retrieval Using Co-Occurrence Data", *Journal of Documentation*, Vol. 34, No. 3, 1978, pages 189-216.

Harper-80 – Harper, D. J., "Relevance Feedback in Automatic Document Retrieval Systems: An Evaluation of Probabilistic Strategies", Doctoral Dissertation, Jesus College, Cambridge, England.

Keen-71 – Keen, E. ”Evaluation Parameters”, in *The SMART Retrieval System - Experiments in Automatic Document Processing, G.* Salton (ed.), Prentice-Hall, Inc., Englewood, New Jersey, 1971, Chapter 5.

Lee-85 – Lee, D. L.,”The Design and Evaluation of a Text Retrieval Machine for Large Databases”, Ph.D. Thesis, University of Toronto, September 1985.

Lehnert-91 – Lehnert, W. and B. Sundheim, “A Performance Evaluation of Text- Analysis Technologies”, *AI Magazine*, Vol. 12, No. 3, Fall 1991, pages 81-93.

Paice-84 – Paice, C. “Soft Evaluation of Boolean Search Queries in Information Retrieval Systems”, *Information Technology*, Research and Development Applications, Vol. 3, No. 1, 1983, pages 33-42.

Saracevic-95 – Saracevic, T., “Evaluation of Evaluation in Information Retrieval”, Proceeding of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1995, pages 138-145.

Veerasamy-96 – Veerasamy, A. and N. Belkin, “Evaluation of a Tool for Information Visualization of Information Retrieval Results”, In Proceedings of the Nineteenth Annual ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, N. Y., 1996, pages 85-93.