Division of Research
Graduate School of Business Administration
The University of Michigan

July 1984

DATA AND DOCUMENT RETRIEVAL

Working Paper No. 382

Michael D. Gordon

Introduction

The purpose of this paper is to compare and contrast data retrieval and document retrieval. Although these two types of retrieval will be better defined, the meaning of each is just what we might expect. That is, data retrieval is the type of retrieval we usually think of as occurring in a network, hierarchic, or relational database. Document retrieval is more akin to the process of looking for library books about some topic and then being presented with a list of pertinent books. More particularly, we shall assume this process is managed by a computer.

Somewhat paradoxically, authors who write about these two types of retrieval seem to fall into two camps. Either they say the two processes are so similar--since they both manage the storage and retrieval of records--that it is not terribly important to distinguish them, or (and this is perhaps more common) they suggest the two processes are almost entirely unalike.

My feeling is that each of these positions is too extreme. To make this point, this paper will be organized as follows: First, a "life cycle" of an information system will be presented. Without great detail, we will examine, in order, each step in this cycle with respect to data retrieval systems. Document retrieval systems will also be examined against the backdrop of this life cycle. But, to keep matters as clear as we can, we will not look at each step of the cycle in order. Instead, we will consider general need; then logical models; then system performance and evaluation. Finally, we will re-examine the entire cycle in order, from a better informed perspective.

Life Cycle of an Information System

A computer information system has as its components computer hardware, system software, application programs, files of data, and even the people who must use the system. The complexity of such systems is so great that they are not just built once and for all. Instead, information systems evolve through a life cylce. This cycle, described in greater detail elsewhere, is comprised of these stages:

1. Statement of general need: the information requirements of an enterprise are stated.

2. Examination of need: the need is studied in greater detail; a feasibility study may be performed.

3. External system design: a logical, user-view design of the system is created.

4. Internal design: the system's internal structure is considered and formulated.

5. Construction: a system meeting external and internal specifications is constructed.

6. Testing/simulation: the system is tested in a real life setting or its performance is simulated.

7. Operation/observation: a large scale operational system has its performance monitored.

8. Maintenance/modification: the system is modified to better suit the purposes for which it was created or is simply maintained in proper working order.

DATA RETRIEVAL (Fact Retrieval)

General need

Databases are constructed to house, and permit the retrieval of, data (or facts). We use the terms data and facts interchangeably, since what we store in the database are facts about the things we are interested in. The stored data should conform to various constraints we may wish to impose. All zipcodes, for instance, should be 5-place integers. No state should have two capital cities. The data should be private, easy to get to when frequently needed, and generally "well-maintained".

Particular needs (examples)

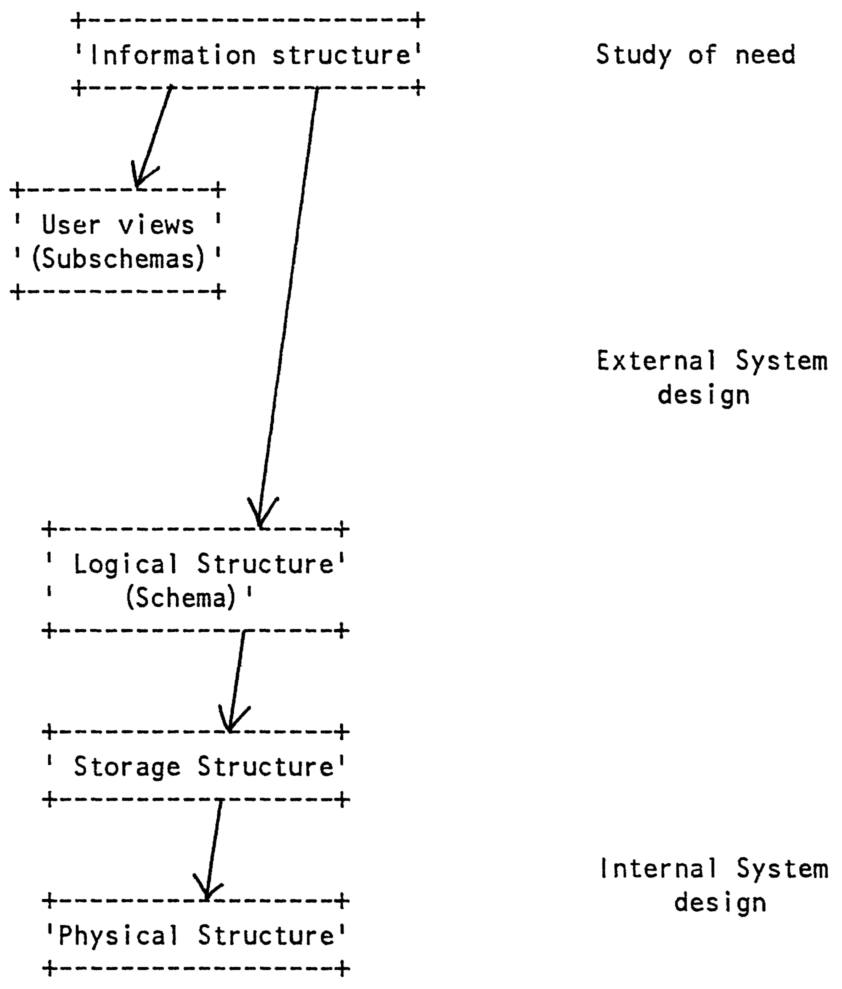
More specifically, fact retrieval may be used in a number of ways, including: pay-roll information for pay-check or tax purposes; a monthly report on inventory; to help book passengers on airlines flights; decision support; or combinations of the above types of uses within one organization.

Modelling within a data retrieval system

The information need of an organization is usually initially modelled independent or any subsequent constraints on the data that may arise due to choice of particular logical model of data or detail of implementation.

This initial pass at representing the data, which produces an information structure, is followed by the modelling of user views (or subschemas); the integration of these views into a single schema; and the implementaion level decisions in which data structures and device details are made.

Schematically, we view this process as something like:



We consider modelling at the information structurelevel first, after which we will look at external and internal system design.

<u>Examination of need:</u> (Information Structure)

At this level we try to determine just what type of information an organization wants to maintain. Basically, two sets of questions arise: questions about the entities (things) we wish to store information about, and questions about possible relationships among various kinds of things.

<u>Entities</u>

What type of objects or events from the real world are we interested in and going to store data about? At this level it would be inappropriate to say we will store data about John Doe; rather, we would say that information about <u>people</u> (a type, in an undefined sense) is to be maintained.

Every type of thing or event should be describable by a collection of pertinent attributes (facts) which characterize for our purposes a member of the type. Name, social security number, and address might be the relevant attibutes we wish to store about any person, for example.

Certain attributes (or sets of attributes) may distinguish objects of the same type. If no social security number is assigned to more than one person, then social security number distinguishes any two people (and so the way they can be accessed in computer storage). Any attribute or atribute set having this property is a candidate key.

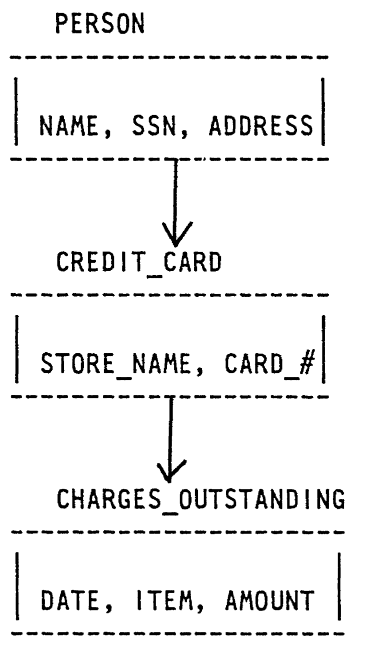
Finally, we may wish to examine the aggregation of attributes we lumped together to see if they might be partitioned to represent several object or event types instead of one. Conceivably, one type of thing under consideration might specify as attribute: name_of_person; last_book_person_read; author_of_that_book; author's_spouse; spouse's_physician; ... It should be apparent that we are trying to include too many unrelated facts in this "mixed" entity type. Semantic guides help us normalize entity-type descriptions so that we do not end up with "entities" which are so all-encompassing.

Relationships

After we have identified the entity types we are interested in, we still have to ask what relationshps exist between or among such entity types. If we have a PERSON entity type and a CREDIT_CARD entity type, an obvious relationship is that of a CREDIT_CARD belonging to a PERSON.

Additional information we need to make explicit about a (binary) relation(ship) is whether or not it is an ("inverse") function (1:M) or a general relation (M:N).

We tie together some of these ideas by producing an entity structure diagram incorporating the entities and relationships we have used as illustrations, plus a new entity type and relationship.



Bachman Diagram;
DBMS-independent model
of data organization

In this diagram we are modelling the fact that a person can hold zero or more credit cards (but no two people hold the same card).

The notational conventions of this Bachman diagram are: entities are represented by labelled boxes with entity attributes contained inside the boxes; entity keys (one of the candidate keys) are underlined; a directed arrow indicates a 1:M relation from the entity type at the foot of the arrow to the entity type at the arrow head.

Logical Design

After the information structure is modelled, we begin to make further models which depend on the database management system we will use to store our data. There are three major "generic" logical models: network, hierarchic, and relational. We consider briefly the network and relational models, since the hierarchic model is more limited in what it is intended to model.

Network Schema

Definition

A Codasyl-type model employs two major logical constructs. The first, record type, names the types of entities our database will consider and lists the attributes which will be associated with each entity type. The other, set type, establishes realtionshps between a particular entity type and one or more other entity types. Particularly, one record type is deemed to be a set-owner; one or more record types are deemed members of the set type. The owner/member distinction becomes meaningful in examining set occurrences. For, in any set occurrence, we have exactly one owner record instance and zero or more member record occurrences which are logically linked to their member record.

We illustrate with pseudo Data Definition Language statements how we might define one of the records and one of the relationships depicted in our information structure diagram:

```
RECORD name is PERSON
     Location...
     Within...
  Person_Name
        Last
        First
  SSN
  Address


SET name is OWNS_CARD
  OWNER is PERSON
      Sorted...
  MEMBER is CREDIT_CARD
```

Pseudo Data Defintion Level Statements

We conclude certain statements with ellipses (...) to indicate that these statements do not really convey logical (non-implementation) information and to remind us that actual Codasyl DDL statements (including those above) do exist which begin defining such storage structure information.

Instantiation

By using data manipulation commands, we represent actual entities and relationships which fit the template of allowable data created by our data definition. That is, whereas data defintion statements tell us what data must "look like" (e.g. that any person is modelled by storing a first and last name, social security number, and address), data occurrences (instances) and set occurrences which conform to such rules are created and stored in the database.

As an illustration, we show a portion of a database conforming to the information structure (and Data Definiton statements) specified:

```
-------------------------------------------
| Jones | Mary | 111-22-3333 | 150 S. Main |
--------------------------------------------<-----+
     |                                            '
  Owns Card                                       '
     |                                            '
     -->----------------       -------------------'
        | Sears | #270 |--> ... | Hudson's | #465 |
        ---------------- <---+  -------------------<----+
          |                  '                 |        '
        Charges_Outstanding  '      Charges_outstanding'
          |                  '                 |        '
        -------------------------              +--------+
        | Jan 15 1975| Shirt| $10|
        -------------------------
```

Comments

1. We are merely indicating that Mary Jones holds two credit cards (two that we have taken the time to draw, anyway). We have drawn ring structures to represent set occurrences (one connecting Jones to her cards; and one for each card and the

items outstanding charged to that card). Codasyl networks do not have to be implemented using ring structures, however. Logically, we require only that we be able to identify any set member with its owner; a set owner with all its members; and each set member with all other members of that set.

2. Record types may be members of more than one set type. Any given set-type occurrence defines a binary relation which is 1:M. That is, each record occurrence of the owner record type has associated with it zero or more record occurrences of the member record type(s). But no record occurrence may have two owners within the same set type.

3. Values are easily supplied to record occurrences. This point may seem obvious. It simply reminds us that once we are given a mold to which data must conform, filling in the values for a particular record occurrence is not subject to debate. In terms of the model we have defined, if we want to represent Mary Jones in our database, values for her first and last names, social security number, and address can be supplied without difficulty: her first name _is_ Mary; last name _is_ Jones; etc. I mention that values are easily supplied in data retrieval not because it is subtle or profound--but as a contrast to document retrieval, where there _is_ considerable difficulty in deciding which values we ought to use in describing a given document.

I make a similar comment now about the relationships we witness in either data- or document retrieval. In data retrieval, ownership of one record occurrence by another is not going to be debated either. (In our example, again, the Mary Jones record is related to (owns) the Sears credit card record with Card_# 270.) In document retrieval, the main inter-document relationships deal with document similarity. And in determining document similarity we cannot simply say yes, document-x is similar to document-y or no, it is not. Our relationships will likely involve <u>degree</u> of similarity. I will describe this distinction more when we discuss document retrieval, particularly clustering. For now, we see that data- and document retrieval differ with regard to this issue.

<u>Relational Schema</u>

<u>Definition</u>

The only logical construct used in the relational model of data is the relation. A relational schema is most easily thought of as an ordered set of attributes (or names of attributes) and a corresponding set of values (a domain) for each attribute.

For illustration, we examine the two relational schemas below:

PERSON(LAST_NAME, FIRST_NAME, <u>SSN,</u> ADDRESS)

CREDIT_CARD(<u>STORE NAME,</u> <u>CARD #,</u> PERSON_SSN*)

The PERSON relational schema is comprised of the attributes

LAST_NAME, FIRST_NAME, SSN (Social Security Number), and ADDRESS. The domain of LAST_NAME is the "set of last names of people." The domain of SSN is the set of all valid Social Security numbers, i.e. all positive, 9-place integers. I underline SSN in this relaional schema to show that it is a key (attribute or set of attributes which distinguishes a unique record occurrence conforming to this schema). (In the CREDIT_CARD schema we see a concatenated key--i.e. set of attributes forming a key.)

In the CREDIT_CARD schema I have also *'d the attribute PERSON_SSN. This was done to point out that this attribute is a foreign key; that is, PERSON_SSN is a key to the PERSON relation. Using the terms "relation" and "tuple", each to be discussed in the next section, I mean by the fact that PERSON_SSN is a foreign key into the PERSON relation that the PERSON_SSN value of a given tuple of any relation over the CREDIT_CARD relational schema picks out a unique tuple in any relation of the PERSON relational schema. Less formally: tell me the PERSON_SSN of some CREDIT_CARD, and I'll tell you <u>the</u> PERSON "associated" with that CREDIT_CARD.

By embedding foreign keys in this way, we can mimic the set ownership between record types in the network model. That is, the foreign key PERSON_SSN embedded in the CREDIT_CARD relational schema serves the purpose (in the way we are using it) of the "OWNS_CARD" set we defined. Because relations, then, serve to define both "things" and "relationships", no other logical construct is necessary.

Instantiation

Using the relational apparatus, we again depict in our database the fact that Mary Jones exists and that she has a Sears and a Hudson's credit card.

Last_Name	First_Name	SSN	Address
.			
.			
Jones	Mary	111-22-3333	150 S. Main
.			
.			

PERSON relation

Store_name	Card_#	Person_SSN
.		
.		
Sears	270	111-22-3333
Hudson's	465	111-22-3333
.		
.		

CREDIT_CARD relation

A tuple is a particular set of values (again, think ordered set of values) meeting the constraints imposed by domains and conforming to the "template" our definition of a schema establishes. <Jones, Mary, 111-22-3333, 150 S. Main> is a tuple "matching" the PERSON relational schema "template." In the same way, we see one tuple each for both of Mary Jones' CREDIT_CARDs.

Collectively, a set of tuples over the same relational schema is a <u>relation</u> over that scheme. The name relation is used because relations (the notion we're trying to define now) are relations (in the mathematical sense; that is a relation is a subset of a Caretesian product, this product taken over the (ordered) set of domains we associate with the (ordered) set of attributes associated to a relational schema). Actually, it is more correct to define tuples and relations without any notion of order.

Like the DBTG model, the relational model is a logical model. It looks here that we have defined a PERSON and CREDIT_CARD relational schema and have produced illustrative relations over each by filling in appropriately labelled flat tables. Doing so is for communications purposes only. It is not necessary to physically represent data in this "table fashion" when we work with the relational data mode.

<u>Storage</u> <u>Structure</u> <u>design:</u> Internal File Structures

Physical files are real collections of records (data) stored in the computer. At the storage structure level of design we discuss the organizations of such files. The nature of the stored data (is it indirect? that is, pointed to, or direct? are the data sorted in some way?), the nature of the sequence of records (do the records follow each other in contiguous storage? or is the record sequence established by pointers?), and the access mechanisms to be employed (how do I get to the record I'm looking for?) are the kinds of questions we examine at this level.

(We still postpone "lower-level" decisions such as data encoding, blocking of logical records into physical records, and choice of physical devices on which to store our data. These kinds of considerations we call physical-design.)

Storage structure is well studied. Some of the file organizations (kinds of physical files) that we consider for use are: heap, hashed, indexed, sequential, hierarchic, inverted, and pointer organizations (and combinations of these).

Construction/Testing/Observation/Tuning/Redoing

Let us suppose now that we have progressed successively through the "need" and desgin stages of our life cycle in designing a data retrieval system. We next build a real system for experimental usage in a chosen setting. Where possible, we may choose to simulate the performance of the system or parts of it.

If all is well (or mostly well) we begin to use our system in the (non-experimental) setting for which it was designed. The performacne of the operational system is then monitored. According to what we learn from doing so, we may have to perform again one of the stages in the life cycle. (Sometimes problems in our system become obvious far before we make our system totally operational. For what has just been said and what I say next this makes no difference.)

The stage we return to depends on the kind of problem we encounter. We may then have to re-do or adjust (tune) parts of the system. At times, it may even become necessary to go back to ascertain needs again. Finding one problem (for example, "need" was not properly ascertained) may make it necessary to perform other steps again as well (logical design, for example is therefore rendered less than adequate, too). Data independence can help insulate us from some of these chain reactions. But that is not at all my main point here. I mean to say only that it will likely become necessary to re-think and/or redesign parts of our system.

The various problems we encounter, being of different natures, arise in conjunction with certain kinds of questions. I illustrate what I mean with a list of problem areas and correspoding questions:

<u>Problem</u>

1. logical: Are all entities and relationships adequately represented?
2. storage structural: Is record access efficient?
3. physical structural: Is it now (or ever) time to allow (actually, cause) data to "migrate" from one medium or device to another?
4. user: Is the system easy to use?
5. control: Are the data private? Are they recoverable?

Loose ends

I bring up another issue which, when what I'm saying is understood, may seem too obvious to mention. Its import is only in comparison to the document retrieval situation where the same cannot be said. The point is: the fact (data) we ask to receive will always be furnished if it is modeled and represented within the system.

This statement is a "data-access" (or, possibly, "data-entry") issue. It says that to someone knowledgeable about a data retrieval system, the data he or she wants will be easy to obtain. For instance, for someone allowed to ask, it is simple to find out Mary Jones' address or which credit cards she owns. For the logical model of data presents us with a picture of what information we can and cannot find in our database and, so, (implicitly) how to access it.

(I'll get ahead of myself enough to say that document retrieval is not like this, however. There is no guarantee that the "right" document will be furnished to an inquirer, even if he/she expresses his or her need reasonably well and this "right document" is indexed and available in the system. Using some document retrieval terminology, we can say that fact retrieval systems are constructed with 100% precision and 100% recall, at least if properly debugged. That is, the deductive, syntactic nature of fact retrieval systems ensures that every time I make a query I will be furnished with all the facts I need and no others.)

The final point I wish to make about data retrieval systems is: other things being equal, performance is measure in terms of space- and time-efficiency. These "other things" can include quite a bit, from ease of use to system security mechanisms. But if these and other such concerns can be factored out, the chief metric of system functioning is efficiency. Again, I suggest what is to follow by stating that, in document retrieval, effectiveness--the percentage of (average) recall and precison of a system--is a more pressing concern than is efficiency. For if the system is not behaving "properly", its speed becomes secondary. And it is a significant task to get document retrieval systems to behave properly.

DOCUMENT RETRIEVAL

General need

Document retrieval systems are built to maintain and provide access to a stored collection of references to information. Computerized library card catalogues are document retrieval systems. The catalogues store pointers to books (information), pointers which will assist library patrons in finding the books that they are looking for.

Particular need (Examples):

Library patrons may have vastly different needs, among them: all legal documents mentioning the name "Miranda"; (some) documents "about DNA"; a (or all) document(s) which will provide the inquirer with _new_ (to him or her) knowledge about computer hardware; a single document which will list the mean monthly temperature in Tucson for the last ten years.

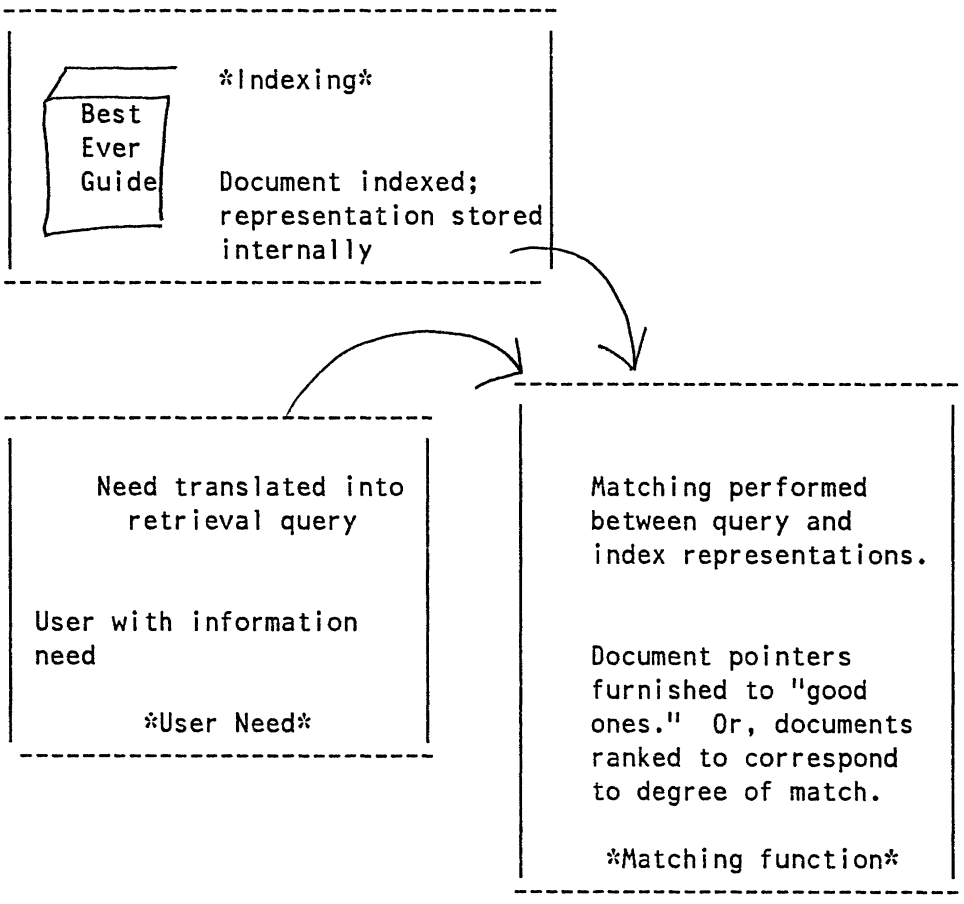
Comments

A "document" means different things in different situations. Most commonly, it means a scientific paper, journal or magazine article, in-house memo, book, or something of the like. But it need not be literature at all (in a very loose sense). A museum which maintains and catalogues references to the items it houses is a "document retrieval system." (The term "information retrieval system" is used synonymously.)

I point out, too, that document retrieval, as I use the term in this paper, will refer to some systems which are actually built and in use; to others that have been implemented on only a very small scale; and even to others which have never made it beyond print.

Process

The document retrieval process consists of three parts.



Indexing

Document indexed; representation stored internally

Need translated into retrieval query

User with information need

User Need

Matching performed between query and index representations.

Document pointers furnished to "good ones." Or, documents ranked to correspond to degree of match.

Matching function

1. Indexing

A document must be labelled or represented in some way. This is the process of "indexing" the document. It may be performed manually (by humans) or with total or partial computer assistance. Indexing sometimes involves selection of descriptive labels from a restricted vocabulary.

Indexing usually is thought of as a process of attaching descriptive labels to documents, but it is conceivable that indexing be done is some totally different way.

Regardless, these document representations are internally stored in the computer, forming the database of the system.

2. User Need

A user with an information need must translate that need into a document retrieval query. The query may be posed in English (natural language), in a more technical "jargon", (by means of a list or vector of terms interpretable by the system), or in a variety of other ways.

3. Matching Function

The query is matched against some or all of the internally stored document representations. In some situations, all documents which match the query at or above a specified matching-threshold are furnished to the inquirer. In other situations, the user may be furnished with a list of documents sorted to correspond to their match with the query. (Actually, a document call number or some other pointer is furnished in either situation.)

Other processes can be seen as participating in some systems as well. First, feedback information may be supplied to a user which causes him or her to submit a revised query. (Other query modification techniques incorporate a user's report of his/her satisfaction with furnished documents to automatically adjust and resubmit a query.) In a different vein altogether, documents themselves may be re-indexed, via feedback, in an effort to achieve better representation.

"Logical" document retrieval models

What does "logical" mean? My classifying a document retrieval model as logical is done in the same spirit that database (fact retrieval) models employ the term--and for the same reasons.

By separating (data retrieval) user views from the total schema--and both of these logical data models from the storage and physical structure of data--we see how independent the levels of data in a datbase are: the concerns we have at one level are divorced for the most part from our concerns at another. And so changes at one data level can be limited to just that level (for the most part).

But this same separation of level of data can, and should, be made in document retrieval, too. As they are designed and conceived today, the logical structure of a document retrieval database is less rich than that in most fact retrieval databases. By using the expression "less rich" I mean that in a document retrieval database we customarily define just one thing: documents. And all of these we describe with the same logical machinery, a weighted index vector, for example. Data retrieval databases usually describe many different types of things, each type being described in a different way. (For instance, PERSON record types and CREDIT_CARD record types are desribed considerably differently.) Adding to the richness of the data-retrieval database are also the relationships that must be described between entity types.

But before we lose track of the main point, I assert that, despite the less rich structure of the document retrieval database (or its model), the same data separation should should be employed in thinking about, designing, and building document retrieval systems. That it is not is seen from authors who speak of designing document retrieval models with the express purpose of building a "faster" one (less time from making the query until the search is completed). These authors confuse the fact that they are addressing storage structural (efficiency) issues--nothing more. But document retrieval systems have their logical machinery to consider as well. In this spirit we use the term logical data models (i.e., independent of storage structural considerations). The examples of models that follow should point up this difference. And to help clinch matters, possible storage strucutres will be described for certain logical models.

Genera of logical models

There are at least two major types of logical document retrieval models, and I will also briefly mention a third type (genre) that seems unlike the first two.

1. Models which emphasize representing individual documents

In such models (notice we are describing a whole class of models) every document is indexed independently of all others. Every document, therefore, receives its own description.

2. Models which emphasize "clusters" of documents

Such models impose logical structure on a set of documents by dividing them into (disjoint or intersecting) document clusters. A cluster then stands as proxy for the set of documents during the matching function process (or its first stage).

3. The third "genre" is actually less a genre than a model itself, described by Doyle, which seems to fit into neither type above. In rough outline, Doyle's model resembles a semantic network, with various kinds of nodes, whose edges are formed according to statistical co-occurrence data.

High level nodes might represent such subject headings as "Biology", "History", "Mathematics", etc. Links (edges) between such nodes represent how related these terms are. So a Biology-Mathematics link might be of strength .25 (in some undefined sense for our purposes), weak perhaps, but considerably stronger that a Biology-History link with strenth .07.

Other nodes represent notions at a more specific level. "Zoology" and "Botany" would likely appear as network nodes, too. Each would be linked with Biology and each other, with measures of their co-occurrence strenth listed, too. (Co-occurrence strength is a measure of the frequency with which two terms co-occur in the literature.)

Overall, we have nodes of many, many different levels of specificity, including (finally) nodes representing individual documents. All are inter-linked and "weighted" (at least those with links of sufficient strength.)

The job of the user is to physically navigate in this network structure using the co-occurrence (strength) cues and his or her own impressions of what topics or notions are alike to guide the way. This navigation would take place in an interactive way, user perhaps sitting at a graphics terminal, light pen in hand, pointing to various nodes, the details of whose inter-node structure becomes evident only when this node is pointed to.

Models which emphasize representing individual documents

I now talk about logical models which rely on representing individual documents as their chief logical device. Seven examples of logical models falling into this category have their workings explained.

A classification scheme

Just as a lack of rigor leads to failure to make a logical versus storage structural distinction in describing document retrieval systems, a similar laxness often prevents some logical models from being fully understood from their descriptions. For any logical model has three parts, and no pair of them even is sufficient to describe a logical model (although authors in the field make this mistake by trying to do so). These three parts are the document representation to be employed within the model; the style of query which is acceptable; and the matching function which relates these other two.

The way I will classify any logical model, then, is to describe it as a triple of this form:
<document rep, query rep, matching function>
("rep" is short for representation).

The examples that follow are not presented as representatives of classes which partition or totally represent logical document retrieval models. They are just examples of models occurring in the literature. In some cases I've made up a name to describe a model.

1. Simple boolean model

<binary doc vector, single query term, "logical" match>

(I will often use the term "doc" for document; "vec" for vector.)

Conceptually, in this model we may imagine a set of documents being represented by a binary matrix, D, as pictured:

$$
\begin{array}{cccccc}
 & \text{term-1} & \text{term-2} & \dots & \text{term-N-1} & \text{term-N} \\
\cdot & & & & & \\
\cdot & & & & & \\
\text{doc-i} & 1 & 0 & \dots & 1 & 1 \\
\cdot & & & & & \\
\cdot & & & & &
\end{array} = D
$$

Document-i is represented by the i-th row in the matrix and D-ij = {1 if doc-i is about term j; 0 otherwise

The "logical" match is an assignment rule stating whether a given document should be presented (or, deemed relevant to) a given query. More precisely, we characterize this assignment rule as follows:

M: Binary_Document_Vectors X Boolean_Queries --> {Yes, No}

by M(b_d_v, b_q) = {yes if b_d_v satisfies b_q;

no otherwise

That is, "logical" match is a function taking an ordered binary document vector-boolean query pair into the values {Yes, No} according to whether the docuement" satisfies" a query.

A query, q, is "satisfied" by a document in the sense easiest illustrated by example.

<u>Example</u>

	term-A	term-B	term-C	term-D
doc-1	1	0	0	0
doc-2	1	1	0	0
doc-3	1	1	1	0

In this example, all three documents satisfy the single (Boolean) query term A since each is indexed as being "about" A. On the other hand, only doc-3 satisfies query C.

2. General boolean model

In this model, we expand the set of valid queries to include any (well-formed) boolean query. (Otherwise, this model is identical to the first.)

A boolean query is any composition of index terms and the logical operators & (AND) V (OR) and ¬(NOT) plus parentheses, as long as it is well-formed.

Example

	term-A	term-B	term-C	term-D
doc-1	1	0	0	1
doc-2	0	1	1	0
doc-3	1	0	1	0

In this example, all documents satisfy the boolean query A V B; none satisfies A & B; only doc-3 satisfies A & C.

Comment

A logically equivalent way of representing a document in the simple or general boolean model is as a list (unordered set) of index terms. In the last example, for instance, doc-1 becomes the set (list) {A, D} under this representation. ("A" and "D" are shorthand for "term-A" and "term-D", respectively.)

Full text retrieval systems logically employ the general boolean model. In such systems a document is represented by (a list of) every substantive word used in that document.

Note we are adopting our alternate document representation in saying this. Additionally, the dimension of vectors in this model (thinking in terms of the original description of the general boolean model again) "grows" as documents containing "new" words are indexed.

Possible storage structure and implementation of match

To reinforce the idea that logical and storage structural considerations are separate, I suggest a likely way for accommodating the storage and matching inherent in the general boolean model.

The method is to represent our document matrix (what I called "D" originally) by inverted lists (sets) and to perform our logical matching by means of set operations.

Example

If D=

	term-A	term-B	term-C	...	term-N
doc-1	0	1	0	...	0
doc-2	1	1	1	...	0
doc-3	0	0	1	...	1

then Set-A (inverted list-A) = {doc-2}, set-B = {doc-1, doc-2}, and so on.

The query (A & B) V C is executed by (Set-A INTERSECT Set-B) UNION Set-C.

(Compare this to doing a full scan over millions of records, most of which are not indexed with any of terms A, B, or C.)

The ¬ operator is implemented by means of set complement.

3. Counting boolean models

<binary doc vec, conjunctive boolean query, "counting measure">

We return to another example of logical model, counting boolean models.

To explain the notion of "counting measure", we resort to our logical representation of documents as sets (lists) of index terms. A conjunctive query, one allowing only conjunction of terms, will also be represented as a set of terms (its conjuncts).

In this way, we can have:

X = set (list) of index terms describing some document

Y = set of terms used in conjunctive query

Some commonly described matching functions are:

a. $\| X \text{ INTERSECT } Y \|$	Simple matching coefficient
b. $\dfrac{2 * \|X \text{ INTERSECT } Y\|}{\|X\| + \|Y\|}$	Dice's coefficient
c. $\dfrac{\| X \text{ INTERSECT } Y \|}{\| X \text{ UNION } Y \|}$	Jacard's coefficient

d. $\dfrac{|X \text{ INTERSECT } Y|}{\min\{|X|, |Y|\}}$ Overlap coefficient

Note that the last three measures are attempts to normalize the effects the underlying cardinalities $|X|$ and $|Y|$ have on the cardinality of $|X \text{ INTERSECT } Y|$.

<u>Coordination level</u>

A coordination level is sometimes advocated for use with the simple matching coefficient. The coordination level supplies a parameter by which we produce a function telling if a document and query do or do no "match" (according to the simple matching coefficient).

That is, we have

$$\text{Match\_level-t}(X,Y) = \{\text{Yes, if } |X \text{ INTERSECT } Y| >= t; \quad \text{No, otherwise}\}$$

Possible storage structure

Again, I illustrate a likely storage structure and implementation, this time for simple matching with coordination.

Our storage structure represents our binary document vectors by inverted lists (sets); our simple match with coordination is implemented by "combinatoric" set operations.

Example

Given a query Q = A & B & C & D, we get coordination_level_4 = (the set of documents with four--or more, but this is impossible) terms in common with the set {A, B, C, D}, i.e. Q)

$$= \bigcap_{i=A,B,C,D} \text{Set-}i \qquad (\text{Set-}i = \text{inverted list-}i)$$

Also, coordination_level_3 (again a set) =

$$\left(\bigcap_{i=A,B,C} \text{Set-}i\right) \cup \left(\bigcap_{i=A,B,D} \text{Set-}i\right) \cup \left(\bigcap_{i=A,C,D} \text{Set-}i\right) \cup \left(\bigcap_{i=B,C,D} \text{Set-}i\right)$$

Generally, then, we have coordination_level_j (for a query with n terms) = Union of $\binom{n}{j}$ sets, each set the intersection of one of the $\binom{n}{j}$ combinations of the n terms.

4. <u>Similarity based vector models</u>

<n-D real doc vec, n-D real query vec, "similarity measure">

(n-D stands for n-dimensional.)

Logical models in the literature sometimes attach real values to document vectors (usually in the interval from 0 to 1, inclusive) in an effort to more precisely describe what a document is "about". Queries, in such models, are described similarly.

<u>Example</u>

	term-A	term-B	...	term-N
doc-x	.4	0	...	.5
query-y	.6	.2	...	.3

A document description of the above sort assigns a value (weight) to term-j of document-i according to "how much document-i is about term-j." Automatic assignment might resemble:

$$\text{weight of term-j in document-i} \propto \frac{\text{\#occurrences in document-i of term-j and near synonyms}}{\text{total \# occurrences of all terms in document-i}}$$

Queries in such a system are "guesses" at how docuements being sought will be indexed.

Similarity measure

The "similarity" between a documnet D = <d-1, ... , d-n> and a query Q = <q-1, ... , q-n> must be calculated.

Example

Salton's Cosine measure calculates the n-dimensional Cosine of D and Q (where D and Q are taken to be vectors in an n-dimensional vector space). That is,

$$\text{Cosine }(D,Q) = \frac{D \cdot Q}{||d|| * ||Q||} = \frac{\sum_i (q\text{-}i * d\text{-}i)}{\left[\sum_i (q\text{-}i)^2 * \sum_i (d\text{-}i)^2 \right]^{1/2}}$$

("." is dot product; * is normal multiplication; ||X|| is the lenght of X, X a vector.)

Comments

We might wonder about such a model:

1. Are the n dimensions truly "orthogonal" (or are they "dependent")?

2. Are the n-dimensions similarly scaled (or are we measuring with something like "feet" in one component, and "inches" in another)?

3. Can we afford the loss of information about the magnitude of

a document vector (which is normalized away in our computation of cosine)?

5. Discrimination-based models

<n-D binary doc vec, n-D real query vec, "weighted sum">

Note: we are employing a new kind of triple. Its use will be explained below.

The idea here is that all terms are not equal. (At least as far as being of value in helping us decide whether a document should be furnished in response to a query.)

Example

I use an extreme example to help explain the above point.

Suppose that term-i is used as an index term in describing 90% of the documents in some document collection. (And so it is not used in only 10% of the collection.) Suppose also that term-j is used as an index term in only 5% of the same collection (and, so, not used in indexing the remaining 95%).

Then, in the spirit of this model, an inquirer wanting a document "about j" should see a very "strong" match between his request and one of the few (5%) documents that is indexed with term-j. (The match should be much stronger than the match between a query "about i" and one of the documents indexed with term i; for, remember, 90% of the documents are indexed with this term.)

Construction of triple

In this sense, j is a better discrimination term than i. And so, we might desire a matching function, M, relating queries and documents such that

1. M = M_a + M_b + ... + M_n; and
2. M_j (Query involving term-j, doc. indexed by term-j) >
   M_i (Query involving term-i, doc. indexed by term-i).

To accomplish this we can represent any user's query by Q = <q-1, ... , q-n> where

q_i = {0 if user not interested in docs "about" i;
otherwise,
a positive real weight reflecting the "discrimination power" of term-i (done in the sense above: the better a discriminator, the higher the weight}

Our matching function, M(document,query) can be a "weighted sum". That is, we can have

M(document,query) = Document . Query = $\sum_{k}$ d-k * q-k

Again, "." is dot prodcut; * is regular multiplication. "Document" and "Query" are a binary document vector and a real query vector, respectively. The latter is constructed in the way just mentioned; the former is just a list (actually) of the terms which the document is about. The resulting calculation, M, becomes, therefore, a "weighted sum."

Comments

1. This model is extendable so that documents, too, can employ real weights. These real weights reflect the extent to which a document is "about" the varous terms used to describe it. The calculation of M remains the same.

2. If the index terms are not independent, we may be either "double-accounting" or "under-accounting" in calculating M. In other words, a more complicated expression than a (linear) weighted sum might best reflect degree of match.

6. Probabilistic Models

<real "aboutness" doc vec, term, probabilistic" relevance">

Note: "term" is a single index term. The other two terms in this triple will be explained below.

Two terms in the document retrieval literautre, often used, but not always in anything more that a "colloquial" way, are: "about" and "relevance". An important contribution of Maron and Kuhns' probabilistic model is that these terms are defined in precise, well-defined ways.

Sample space

The model, being probabilistic in nature, relies on a sample space. This sample space is the set of ordered pairs

{<q,d>|q=single index term used to pose a query;

d= document found relevant by inquirer to that query}

For example, the outcome (sample point) <A, 465> refers to document 465 being furnished to, and then found relevant by, an inquirer in response to his/her query, "A".

Events

The events we need to consider to understand the model are:

D-i = the event that doc-i is found relevant (no matter what the request) ={<q',d'>|d' = doc-i}

I-j = the event that a request is made with (index term) j
in posing a query $=\{<q',d'>|q'=j\}$

Definitions

Maron and Kuhns use the events above in defining what "about" and "relevance" are to mean.

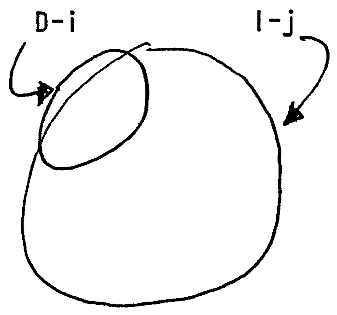
1. The extent to which doc-i is <u>about</u> term-j
=(def) $P(I\text{-}j|D\text{-}i)$
2. The relevance of doc-i to term-j =(def) $P(D\text{-}i|I\text{-}j)$

We may easily imagine less formal ways to bring life to the definitons.

1. "About": Of a "totally representative" group of users, all of whom found doc-i relevant to their information need, we ask what percentage requested this document with query term-j. This percentage corresponds to how much doc-i is about index term-j.
2. "Relevance": Oppositely, of a "totally representative" group of users making a request with index term-j, we ask what percentage found doc-i relevant. This percentage corresponds to the relevance of the document to the query term.

Example

A diagram, assumed to be drawn so that areas of events reflect their probabilities, illustrates better the distinciton:



In this diagram, doc-i is very much <u>about</u> term-j; however, doc-i is <u>not</u> too <u>relevant</u> to term-j.

<u>Relation between about and relevance</u>

We use probability calculus to show the relation between "about"-ness and relevance; in doing so we will see the intended application of the model.

1. $P(I\text{-}j \,\&\, D\text{-}i) = P(I\text{-}j \,\&\, D\text{-}i)$ ==>

2. $P(D\text{-}i \mid I\text{-}j) * P(I\text{-}j) = P(I\text{-}j \mid D\text{-}i) * P(D\text{-}i)$ ==>

3. $$P(D\text{-}i \mid I\text{-}j) = \frac{P(I\text{-}j \mid D\text{-}i) * P(D\text{-}i)}{P(I\text{-}j)}$$

The left hand side of line 3 is exactly what Maron and Kuhns define as the relevance of document-i to term-j. The conditional probability in the right hand side is how "about" I-j D-i is.

Now, the goal of a document retrieval system is to take user queries and provide useful (hopefully) documents. In particular, in response to query j, we may consider one document more relevant than another in the exact sense Maron and Kuhns define.

But, in looking at line 3, we see that the denominator of the right hand side is constant for fixed j no matter what i is. And so, we may eliminate this from our relevance calculation, obtaining:

4. $P(D\text{-}i \mid I\text{-}j) \propto P(I\text{-}j \mid D\text{-}i) * P(D\text{-}i)$

This last equation expresses the spirit of Maron and Kunhs' model. The left-most term, $P(D\text{-}i \mid I\text{-}j)$ represents the "proabilistic relevance" of a document to a query. This term, calculated from the right hand side, thus embodies the third component of our descriptive triple, the matching function.

The conditional probability on the right hand side of the proportion (that is, $P(I\text{-}j \mid D\text{-}i)$) is, again, an "aboutness" term. Any given document, Document-i, is described, at index time, with an aboutness vector, whose value at the j-th position is the aboutness term, $P(I\text{-}j \mid D\text{-}i)$. The remaining term in line 3, $P(D\text{-}i)$, (or the probability that document-i is found relevant to any request), is discernable from the circulation data in a library.

Reiterating, we have

$$\underbrace{P(D\text{-}i \mid I\text{-}j)}_{\substack{\text{\underline{relevance} match:} \\ \text{derived from r.h.s.}}} \quad \propto \quad \underbrace{P(I\text{-}j \mid D\text{-}i)}_{\substack{\text{supplied at index} \\ \text{time in forming} \\ \text{\underline{aboutness} vector}}} \quad * \quad \underbrace{P(D\text{-}i)}_{\substack{\text{"collectable" from} \\ \text{library statistics}}}$$

Since we have "cancelled out" a constant term in producing line 4 from line 3, the relevance match (the left hand side of the proportion in line 4) is not precisley Maron and Kuhns' relevance of a document to a query, but is in proportion to it. And clearly a rank ordering of document to a query according to either relevance of "relevance match" will be identical. Maron and Kuhns in fact suggest that the function of a retrieval system ought to be producing such rankings. The user, then, may be assured of seeing the "most relevant document the system can produce" first; then the next most relevant; then the next; and so on. This scheme lets the user stop his or her search as soon as his/her need has been met instead of presenting him/her with an unordered document set which must be searched through to see if any satisfies his/her need.

7. Another probabilistic model

<binary document vector, *, "probabilistic weighting">

Probabilistic models are often discussed currently in the document retrieval literature. These models differ from that suggested by Maron and Kuhns. Here is another, discussed by vanRijsbergen, Croft, and others, which attempts to answer a fundamental retrieval question: given a document's description, how likely is the document to be relevant to an information need expressed by a given query? Note that this model tries to answer the question of a document's relevance to a query by making a calculation from its description, a strategy not employed in Maron and Kuhns' model. Note, too, that the model does not rely on a particular type of query, a`fact I have indicated by "*" in the second positon of the descriptive triple above.

Sample space

All points (and so events) in the sample space are with respect to a given query, q, which does not figure directly into the calculations in the model. But I will use notation in describing events to emphasize that the model relies on answering the question of relevance with respect to this query.

A sample point in the model is of the form:
<document, binary document description, relevance>
where "document" is some document (its title, if you will); "binary description" the way it is described; and "relevance"

one of the values "True" or "False" according to whether or not the document is relevant to the query, q.

Events

The events we need to describe to understand the model are these:

1. q-rel-i= the event that doc-i is relevant to the query, q = {<doc', bin_desc', rel_val'>|doc'=doc-i & rel_val=true}

2. ¬q-rel-i= the event that doc-i is not relevant to the query, q =
{<doc', bin_desc', rel_val'>|doc'=doc-i, rel_val=false}

3. assigment-i-X: event that doc-i has the binary vector, X, as its description. That is, X=<x1, ... , xN>, (the xi's are fixed; xi either 1 or 0 for 1 <= i <= N) is the way doc-i has been indexed, and assignment-i-X=
{<doc', bin_desc', rel_val>|doc'=doc-i, bin_desc'=X}

Fundamental issue

There is a question which all this probabilistic machinery is constructed to answer. In informal terms, the question is: Given a document's description, is it more likely the document is relevant (to the underlying query, q) or not relevant to it?

We express this question symbolically by:

P(q-rel-i|assignmnet-i-X) > P(¬q-rel-i|assignmen-i-X) ?

By some simple algebra, we may phrase this question in an equivalent way:

(1/P(assignment-i-X))* P(assignment-i-X|q-rel-i) * P(q-rel-i) >

(1/P(assignemnt-i-X))* P(assignemnt-i-X|¬q-rel-i) * P(¬q-rel-i)?

Assumption

A calculation will be performed to answer the above (second) inequality. But some major assumptions are made to carry out the calculaton.

The first assumption is that, given it is known a document is relevant to a query, the probability that the document has a given assignment, X, can be broken into a product of N (the number of positions in X) conditionally independent probabilites. (That is, if we conseder X a collection of indicators, X1, ... Xn, then the expectation of each Xi is conditionally independent given q-rel-i). In symbols, this amounts to:

$$P(\text{assignment-i-X}|\text{q-rel-i}) = \prod_{j} P(\text{assignment-i-Xj=Xij}|\text{q-rel-i})$$

(where X=<X1, ... , Xn> is an N-place binary vector; and assignemnt-i-Xj=Xij is the event that the j-th position of X (which describes doc-i) is Xij).

The second assumption, totally analagous to the first, is that, as indicators, X1, ... , Xn are conditionally independent of ¬q-rel-i. Or:

P(assingment-i-X|¬q-rel-i) = P(assignment-i-Xj=Xij|¬q-rel-i)

Calculation

With these assumptions, and using the notation

p-k = P(X-k=1|q-rel-i) and

q-k = P(X-k=1|¬q-rel-i)

we may compare the logarithms of both sides of the inequality we are interest in (the second one). Upon subtracting (log of right hand side from log of left hand side), we obtain:

If g(X) = $\prod_{k}$ c-k * x-k + C > 0

then document-i is relevant to query q

otherwise, it is not. (At least this is the way our calculation tells us to make a relevance judgment.)

In the above expression,

c-k = log((p-k/(1 - p-k))/(q-k /(1 - q-k)))

C = constant for any document which will depend only on the query being made.

Comments

1. If the p-k's and q-k's (which, I emphasize, are with respect to a given query, q) are known, then the c-k's are easy to calculate; and so the calculation of g(X) is an easy, linear sum of certain c-k's (the ones asssociated with document description positions which are 1's, not 0's).

2. We can make an intuitive interpretation of c-k by looking at

it as a log (which let's now forget about) of a ratio. The numerator of this ratio is p-k/(1 - p-k).
The denominator is q-k/(1 - q-k).

But the numerator is a ratio itself, this ratio contrasting (dividing) the probability that, given doc-i is relevant to q, doc-i is indexed with x-k with (by) the proability that it does not.

The denominator (of the entire ratio) is again a ratio, just like the one I've just tried to describe, but with respect to the fact that document is not relevant to q.

So c-k =

$$\underbrace{\frac{P(x\text{-}k=1 \mid q\text{-}rel\text{-}i)}{P(x\text{-}k=0 \mid q\text{-}rel\text{-}1)}} \Bigg/ \underbrace{\frac{P(x\text{-}k=1 \mid \neg q\text{-}rel\text{-}i)}{P(x\text{-}k=0 \mid \neg q\text{-}rel\text{-}i)}}$$

which suggests:

$$\frac{\text{relative likeliness of x-k being 1 (given doc relevant to q)}}{} \Bigg/ \text{relative likeliness x-k being 1 (given doc not relevant to q)}$$

In this fashion, the entire ratio, (c-k), can be thought of as a measure of the discriminatory ability of x-k.

3. The independence assumptions are quite big. For, given we know a document is relevant to a query, the assignment of various terms (x-i's) will likely be <u>quite</u> <u>dependent</u> on each other.

Cluster based methods

Purpose

A document collection which is not partitioned or divided in some way may necessitate the computation of a match between a given query and every document index in a document retrieval database. With only a moderately large database, such searches can begin to take far too long; with very large databases, such "serial-matching" techniques become prohibitively lengthy (especially for on line retrieval).

As a result, documents are sometimes grouped into clusters (sets). In response to a given query, an entire cluster may be presented. Alternatively, in response to a query, one or more clusters may be selected and serially-searched by the system before presenting documents to a user.

Dissimilarity-based models

Some cluster-based methods rely on pair-wise dissimilarity measures. The dissimilarity between two documents is a real value between 0 (totally similar) and 1 (totally dissimilar). Dissimilarity is a symmetric measure (dissim(doc-1,doc-2) = dissim(doc-2,doc-1)). Various statistical measures are suitable as dissimilarity measures.

I present first the single-link clustering model. It is based on a dissimilarity matrix, which it uses as input, and produces a dendrogram (a tree with levels) as output.

Clusters do not have to be disjoint; also, some clustering methods do not rely on a dissimilarity matrix. A non-hierarchic, intersecting cluster model is presented second.

<u>Example:</u> single-link

doc	1	2	3	4	5	6
1	0	.2	.3	.5	.5	.5
2		0	.3	.5	.5	.5
3			0	.5	.5	.5
4				0	.1	.4
5					0	.5
6						0

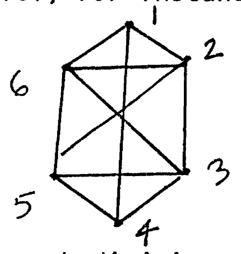
symmetric dissimilarity matrix

We use this matrix to form clusters (which I will show by graphs) at various "levels". Using the levels, .1, .2, .3, .4, and .5 (the positive values found in the table) we form a graph at the t-level as follows:

There is a unique node for each document.

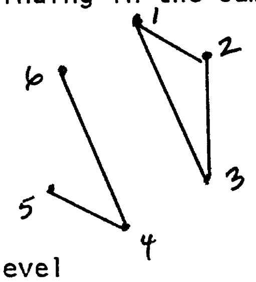
Two nodes are connected if and only if the dissimilarity between them is at most t.

At the .5-level, for instance, we obtain:



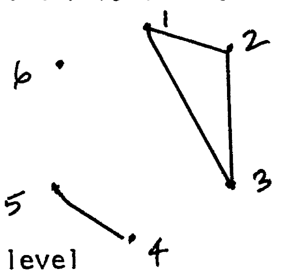
We consider each disjoint subgraph of any graph a cluster. (At the .5 level there is only one cluster then.) In this way we are partitioning the set of documents. Our first partion is {{1,2,3,4,5,6}}.

Continuing in the same way we obtain:



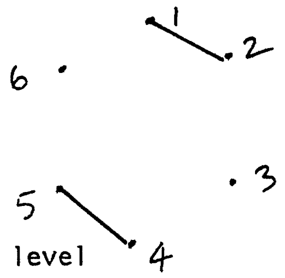
.4 level

{ {1,2,3}, {4,5,6}}



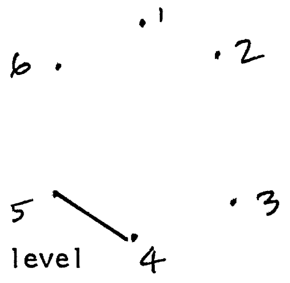
.3 level

{ {1,2,3}, {4,5}, {6}}



.2 level

{ {1,2}, {3}, {4,5}, {6}}



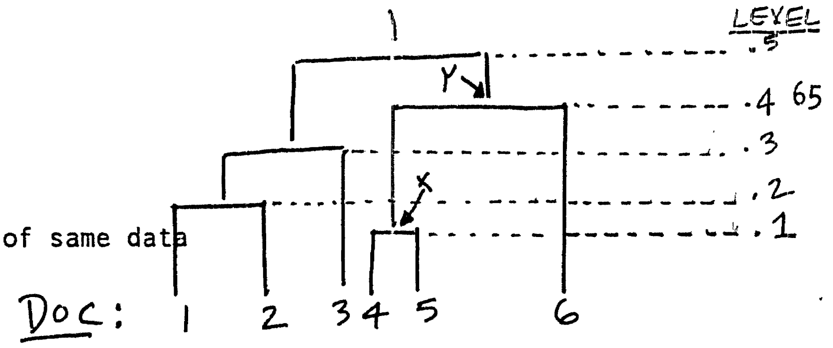
.1 level

{ {1}, {2}, {3}, {4,5}, {6}}

Dendrograms and cluster representatives

Another way of depicting these same data is with a dendrogram (tree with levels). The dendrogram for the cluster in the example is:

Dendrogram of same data



Each node in the dendrogram represents a cluster (set of documents). X, for example, stands for the cluster {4,5}. Note that clusters at a lower level are subsets of clusters at a higher level. $X \subseteq Y$, for example. Each individual document is a leaf of the dendrogram.

A method is needed to represent clusters. Two approaches (there are others) are:

1. maximally linked representative

Re-examine the graphical representation at the .4 level. There we have 2 disjoint subgraphs (2 clusters). In the one, Y={4,5,6}, we see that 4 has 2 links emenating from it whereas 5 has only 1 and so does 6. 4 is thus the "maximallly linked representative" of this cluster (there may be more than one) and is used to describe the cluster as a whole.

2. "Average" representaive

Suppose documents are represented by real vectors. The representative of a cluster containing three documents would be their vector sum divided by three. Similar techniques apply to clusters of binary vectors.

Search Strategies

Two types of searches can be used with dendrograms, top-down and bottom-up.

Top-down

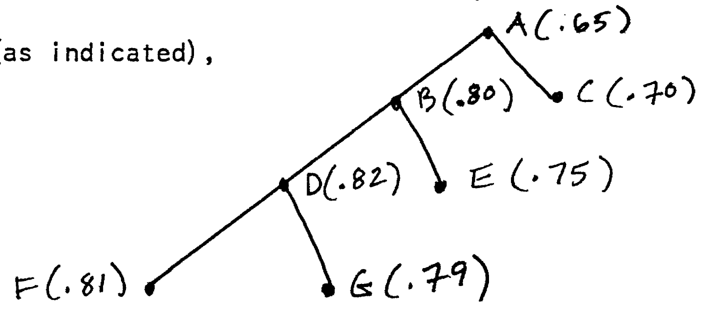
Visit first the root of the dendrogram (hereafter considered an ordinary tree) and match the query with the cluster representative of this node (the root).

IF the node being visited has children (direct descendants) and any of them better matches the query than does the node being visited

THEN visit the child which best matches the query and apply (recursively) the same test as above (the "IF" part) with that child

ELSE /* no children or none which better matches than the node being visited */
node being visited is the desired node (desired cluster).

For a given query, q, and a dendrogram (drawn as a tree) with nodes A, B, ... , G which match q with the associated real values (as indicated),



the pattern of visitation is A, B, D. And D is taken as the desired cluster.

Bottom-up

Bottom up strategies rely on the user knowing one "good" document in the first place. The idea is to provide the user with documents which are most similar to this one--and as many as the user wants.

The flavor of a bottom-up search is given by:

1. Visit first a leaf (the one corresponding to the "good" document already known).
2. Visit the parent of the node being visited provided it contains not more than m documents (where m is a user-supplied parameter).
3. Re-apply step 2 (recursively).

The final node visited has the property of containing the known document, being at most size m, and containing (hopefully) those documents which are most like the "good one."

Algorithmically defined clustering

The clustering method described above relied on pairwise dissimilarity values computed over all document pairs. Another style of clustering does not rely on such a calculation.

Further, not all clustering methods produce hierarchic--or even disjoint--clusters. I present a version of clustering by means of a single-pass algorithm. This method is an example of non-disjoint cluster generation without the aid of a dissimilarity matrix computation.

Example Single pass clustering

```
List all documents docl, ... , docN.
docl is a cluster (and its representative)
For each remaining document, doc-i, do:
    IF doc-i "matches sufficiently" any existing clusters
     THEN add doc-i to that/those cluster(s) (and periodically
re-calculate cluster representatives, too)
    ELSE doc-i becomes a cluster (and its representative).
```

The above algorithm is usually adjusted, too, so that no cluster ever exceeds a certain size. Note that the clusters generated by this algorithm must be serially searched, since there is no structure among them. (They are not hierarchic, for example.)

Competing motivations in clustering

Two competing issues come to bear in clustering: speed of calculation versus theoretical soundness. Single pass clustering, avoiding the calculation of a dissimilarity matrix, is a faster technique than sinle-link clustering(O(nlog(n) v.

O(n*n)). It fails, however, to possess two "theoretical" properties considered quite desirable. First, the order in which the documents are presented determines the resulting clustering. Second, and somewhat related to the first, as the collection size grows significantly, the original clustering may prove to be unsuitable and may need to be totally redone. That is, algorithmic-based schemes most often do <u>not</u> possess this property:

Clustering of (Initially clustered set + many documents subsequently added to the document collection) =

Clustering of (Initial set (unclustered) + [same as above] additional set of documents)

This property is usually achieved only when some underlying theoretical principle (such as pair-wise dissimilarity) regulates clustering.

As such, clustering methods fall into two broad schemes: those which opt for theoretical soundness above all; and those which opt for efficiency first.

Product of search

I have already suggested that there are two different styles of presenting documents to a user. Briefly, I will discuss each.

1. Partitioning

Systems sometimes partition a document collection into two sets with respect to a query: a relevant set and a non-relevant set. The relevant set is really

$\{doc \mid M(query,doc) >= threshold\}$

where "query" is the user's query and M is the matching function being employed.

Even logical models which do not readily fit the formal characterization above are still partitioning in the same sense. I am thinking of the general boolean model which explicitly divides a set of documents into those that "satisfy" a query and those which do not. With just a slight bit of imagination we can change our explanation of that model and devise a threshold so that only documents which match a boolean query at or above the threshold are the "right ones" anyways.

2. Rank order

Instead of partitioning the document set into the "good ones" and the "bad ones", documents may be rank ordered according to how well they match the given query. That is, with respect to query q, doc-i precedes doc-j in the rank order exactly when M(doc-i,q) > M(doc-j,q) (let's disregard ties). The user customarily continues reviewing the documents, in order, until his/her need is satisfied. Note, of course, that we could easily partition a rank-ordered set by saying that the first m belong in the relevant set while the others belong in the non-relevant set.

Evaluation

Retrieval techniques are evaluated to determine the effectiveness of document retrieval systems--sometimes one system compared to another.

The effectiveness measures most commonly discussed in the literature are those which pertain to searches which partition documents--not rank them--and the most common of these I mention here.

The easiest way to explain these two measures is by means of a 2 by 2 table, as below:

	Retrieved	Not Retrieved
relevant*	a	b
not relevant	c	d

Performance of document retrieval system for one query
(* User determined)

Using this table we have:

Recall= proportion of relevant documents retrieved = a/(a+b).

Precision= proportion of retrieved documents which are relevant = a/(a+c)

Note the change in the way we are using the term "relevant" in the table (and also in the definitions). "Relevant" now means "<u>actually</u> relevant to a user's need, as the <u>user</u> determines it." Algorithms which partion documents into a "relevant" and "non-relevant" set are utilizing quite a different interpretation of the term relevant--one that is algorithmic or syntactic in its very nature. It is this difference we are analyzing in our 2 by 2 table (and our definitions of recall and precision). The "relevant" and "non-relevant" sets of documents--according to the system which prodcued our table--are the documents tallied in the left column and right column, respectively; (that is, what we are calling the "retrieved" and "not-retrieved" documents. In fact, recall and precision are statistics which attempt to measure these two very different meanings of relevance: recall measuring the extent to which the "actually relevant" documents were judged by the system to be relevant; precision measuring the extent to which only those documents which are "actually relevant" were judged so by the system.

Aside

I present a 2 by 2 table reflecting the behavior of a data-retrieval system (one that is behaving properly):

	Retrieved	Not Retrieved	
relevant*	x	0	x and y are non-negative integers
not relevant	0	y	

Performance of document retrieval system for one query
(* User determined)

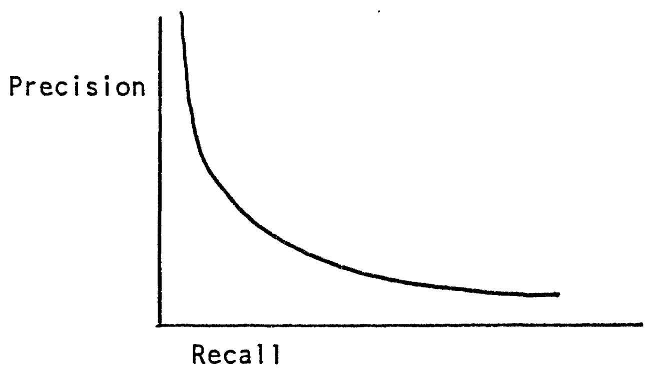
For instance, a data-retrieval request to fetch all records "in which the last name is Smith" will furnish all and only the appropriate records.

The striking difference between the two tables suggests the contrasting nature of the two types of retrieval. The explanation of this difference stems from the inferential nature of document retrieval. That is, since our document descriptions only suggest what a document is about, and since a query only suggests what a user's information need is, a document retrieval system may do no better than infer that doc-x is (is not) useful (relevant) to query-y. And this inference is conducted by means of a matching funciton (i.e., an algorithm) which cannot account for all the indeterminacy inherent in the system.

In data retrieval, on the other hand, facts (or data) "are what they are." There is no need, that is, to resort to a higher level of description to represent them. (My explanation below will help explain the meaning of "higher level.") And users are permitted to ask for only those data precisely represented in the database. Therefore, the "matching function" in data retrieval is tantamount to selecting the best access path. Thus we see the 2 by 2 table of the second form in data retrieval, one reflecting perfect recall and perfect precision.

Think for a moment of using a data retrieval system in document retrieval fashion (that is, in a "higher level" fashion than it was designed for). What sort of precision and recall would we expect if we posed to a database containing the relation
PEOPLE(Name, Salary, Make_of_car, Address)
the query: Which people are doctors?

It is customary to plot recall versus precision for a document retrieval system. In doing so, we usually see a curve resembling the one below, where each point represents system effectiveness for a single query. (Alternately, we observe a similar curve for a system which is furnishing documents in response to a fixed query with some system parameter being changed.)



As the figure suggests, it is easy to achieve high precision or high recall--but usually one at the expense of the other.

Loose ends

Actually, many questions revolving around system effectiveness remain to be answered, including:

1. How should recall be estimated? (Observe that recall involves the number of relevant, but unretrieved, documents--a quantity which can usually be just guessed at.

2. Can a <Precision, Recall> pair be collapsed meaningfully into a single figure? If so, retrieval effectiveness comparisons become easier to make.

3. Are precision and recall <u>adequate</u> effectiveness measures? (There are others.)

4. How can individual preferences be incorporated into measuring system effectiveness? Users do have various needs, after all, from searching for a highly specific document (whose title is forgotten) to doing a very broad bibliographic search. Users vary also in their "futility points" (the maximum number of documents which a user is willing to examine in response to a given query). Issues such as these deserve attention in measuring effectiveness.

5. How effective are document retrieval systems? Surprisingly, in a situation seeminlgly quite conducive to achieving high recall (full text retrieval) a study suggests recall is on the order of only 20%! The implication in that finding is that

natural language is so fraught with "indeterminacy" (alternative expressions for the same idea and the same expression for different ideas) that it is a less than adequate device for document retrieval (at least as it is used in most systems). A caution however: the low recall can be attributed to users' reluctance to use much more <u>general</u> terms in posing a query. For general requsts, while improving recall, can drastically reduce precision (and so present the requester with an overwhelming number of documents).

Re-evaluation of document retrieval

As I indicated at the outset, certain issues in document retrieval will be considered now that the basic problems have been described.

Information structure

In brief, users' needs have not received wide attention in studying document retrieval.

At the earliest stages of designing a system, these needs should be ascertained. Is the "average" user more likely to want all documents on a topic, just one, or something in between? What "futility points" (number of documents a user is willing to look through) does the "average" user have? Are user requesting patterns and library indexing procedures coordinated? (A situation like that in document retrieval exhibits what can happen with poor coordination. At the Smithsonian Institute, geological artifacts are catalogued and may be examined by patrons. Unfortunately, patrons most commonly wanted to make references about artifacts according to their geographic positions, and the artifacts were not "indexed" in this way!)

We have considered various logical models. Various user needs have been indicated, too. A study matching user needs with logical models is desirable in bridging the gap between the two. Scientists, for example, may have information needs highly suited to full text retrieval. Then again, they may not.

Another idea is to build various "record types" within a logical model. For example, we can imagine a document retrieval system with a "scientific record type" and a "fiction record type" (as below) without any difficulty:

```
FIELD    MONOGRAPH//TUTORIAL    THEORETICAL//EXPERIMENTAL
-----------------------------------------------------------
|      |                      |                          | ...
-----------------------------------------------------------
     scientific record type

AGE_LEVEL     MYSTERY//HISTORICAL//...
-----------------------------------------
|          |                           | ...
------------------------------------------
     fiction record type
```

Another information structure issue deals with establishing appropriate "contextual clues." Up to now, we have been interested in establishing only the <u>content-description</u> of a document, (what a document is about). "Contextual clues" are such things as date of publication, author, number of pages, publisher, etc. All of these, in combination, can serve to effectively partition a database into many smaller--and more easily searched--ones.

Logical models

Even when we have chosen a logical model our problems are not over. (Let us suppose that model will represent documents with binary vectors, for example.) We still have other difficult, and still logical, problems to consider, including:

Definition level

How many "dimension" (places) should we use in our document vectors? What terms should we use (how general or specific should they be)? Answers to these questions will strongly influence system effectiveness. (If we need to distinguish scientific from non-scientific documents, then choosing "science" as one of our vector terms makes sense.)

Instantiation

Even when the terms in the model are selected, the problem of deciding whether a document is or is not about that term remains. (I am still supposing that we are using a binary document vector. Similar arguments can be advanced for other models, however.) Even indexers disagree considerably about how to supply values. Quite importantly, too exhaustive indexing (lots of 1's in a binary document vector) promotes recall and hinders precision.

Ultimately, we must determine how the definition-level decisions and the way we instantiate a document (represent it in the model) affect retrieval effectiveness.

Two final considerations:

1. What other models might be employed? (Doyle's model is certainly different fundamentally from most of the other models I have discussed. Other "radical" models can of course be developed.) Possibilities exist for developing systems whose recall and precision is far superior to the existing models.
2. Can't various users conceive of the same database in different ways (much like different user subschemas--or views--make a fact retrieval database appear different to different users)? This "view"-approach would be an attempt to match user needs with logical models. In doing so, a new issue arises: how can these views be integrated (into a single "schema)?

Summary

Perhaps P. Leslie described the overall problem of indexing best:

"Somebody has defined indexing as a game involving two players--an indexer and a user. In this game, the first player (the indexer) tries to guess where the user will look for a particular record. The second player (the user) tries to guess where the indexer put it. The game gets a little complicated when the user tries to guess where the indexer guessed the user would guess the indexer guessed the user would look for it."

Storage Structure and Implementation

We have seen how certain logical models might be implemented. (Boolean models, I suggested, employ inverted lists and set operations. Clustered models likely employ pointer organizations.) But, for others, questions remain as to how data should be stored and the matching function conducted.

For instance, similarity based real models or probabilistic models, to mention just two, need to employ storage and matching techniques which avoid record by record serial searching while still carrying out the logical functioning of the model.

Performance

I break up performance into two categories: effectiveness and efficiency.

Effectiveness

By effectiveness we mean how well do user information needs get satisfied (speed aside). Admittedly, this is a bit vague, since effectiveness can be measured in terms of precision, recall, or a variety of other ways. I have suggested, too, work has still to be done in establishing and validating useful effectiveness measures.

Other issues which require attention are:

1. How effective are the models described in this paper (and others)?
2. How can we correctly attribute effectivenss? That is, I have tried to show that in a document retrieval setting we pass through several stages: selection of a model (boolean v. probabilistic v. ...); selection of "attributes" of a model (which terms will we index with?); and, finallly, instantiation within the model (is doc-i about term-j or isn't it?). Consequently, the decisions we make at stages two and three strongly determine how effective we judge a model.
3. What measures of effectiveness are appropriate for ranked-output systems?
4. How effective do systems remain after considerable update

activity takes place (especially insertions of records)? I have never seen this issue brought up outside the context of cluster-based systems.

Efficiency

Several questions need answering:

1. What amount of storage is needed to implement a model? Can this storage space be reduced somehow? (For instance, can we collapse "sparse vectors", i.e., ones with lots of 0's, in the same way we store sparse matrices?
2. How much time does it take to implement the matching function of model?

Summary

I summarize the chief similarities and differences between data and document retrieval.

Similarities

1. Both obey the same information system life cycle.
2. Each is a repository for, and supplies access to, a vast amount of information.
3. Possibly, they may share logical models. (Suggestions to use a relational data model for document retrieval have advanced.)
4. The underlying storage structures and access mechanisms are the same in both cases.
5. Each type of retrieval should depend on coordinated, centralized administration.

Differences

	Data	Document
User need	Answer to specific question. Assemble, maintain, and provide factual information	Point the way to information
Items wanted	Only records exactly corresponding to query	Items most relevant to query (one, some, all)
Matching	Exact match (between query and record)	Best match (hopeful match!)
Ease of Representation	High	Lower
	Definition level: Important attributes of an entity type are identifiable	Definition level: Important terms are less easily identified (due to issues such as term generality/specificity, exhaustivity and possible effects on matching function)
	Instantiation: Easy to represent fact that particular occurrence of an entity type has particular value for a given attribute	Instantiation: Not "rote." When does D-ij in a binary document vector e.g.? What does D-ij=0.7 in a "real" document vector mean? What are the implications of these values on the matching function?
Performance	Chief concern: efficiency	Chief concern: effectiveness

Essential Bibliography

Date, C. J.
An Introduction to Database Systems, Third Edition,
Addison-Wesley Publishing Company
Reading, Mass., 1981

Maron, M. E., and Kuhns, J. L.
"On relevance, probabilistic indexing, and information retrieval,"
JACM, 3, 1960

Salton, Gerald
Dynamic Information and Library Processing,"
Prentice Hall, Englewood Cliffs, New Jersey, 1975

VanRijsbergen, C. J.
Information Retrieval,
Butterworth and Co., Ltd., London, 1979