REFERENCE RETRIEVAL BASED ON USER INDUCED DYNAMIC CLUSTERING

Robert N. Oddy

Ph.D. Thesis

University of Newcastle upon Tyne, December 1974

ACKNOWLEDGMENTS

My chief debt is to Miss E.D.Barraclough, who has been always ready to discuss the work, contributing many stimulating ideas, and to provide encouragement. Next, I should like to thank the staff and graduate students of the Computing Laboratory; particularly Professor E.S. Page, who gave me the opportunity to do the work, and members of the OSTI-supported Medusa on-line information retrieval project, who made data available for my experiments.

I wish to thank those members of the staff of the University Library who have an active interest in library automation for numerous instructive and provocative conversations. I have also benefited from contact with many information and library specialists in other institutions. I gratefully acknowledge the financial support that I have received from the Office for Scientific and Technical Information (OSTI).

My family have been far more patient and encouraging than I had a right to expect and I am indebted to them all, particularly my wife and children.

R.N.Oddy

ABSTRACT

The problem of mechanically retrieving references to documents, as a first step to fulfilling the information need of a researcher, is tackled through the design of an interactive computer program. A view of reference retrieval is presented which embraces the browsing activity. In fact, browsing is considered important and regarded as ubiquitous. Thus, for successful retrieval (in many circumstances), a device which permits conversation is needed. Approaches to automatic (delegated) retrieval are surveyed, as are on-line systems which support interaction. This type of interaction usually consists of iteration, under the user's control, in the query formulation process.

A program has been constructed to try out another approach to man-machine dialogue in this field. The machine builds a model of the user's interest, and chooses references for display according to its current state. The model is expressed in terms of the program's knowledge of the literature of the field, namely a network of references and associated subject descriptors, authors and any other entity of potential interest. The user need not formulate a query - the model varies as a consequence of his reactions to references shown to him. The model can be regarded as a binary classification induced by the user's messages.

The program has been used experimentally with a small collection of references and the structured vocabulary from the Medlars system. A brief account of the program design methodology is also given.

CONTENTS

Chapter 1

INTRODUCTION

Retrieving references to books, papers, reports, and all the other forms of documentation is part of the job of a library system: prerequisite, in fact, to delivering the actual books, or documents, to the reader. It is a task that may be performed, partly or in whole, by the library user himself, and its nature will depend upon the requirement which prompted him to go to the library, and the type of tools provided for this purpose. We shall be discussing such a tool - an interactive computer program - in the light of our view of the underlying problems of reference retrieval.

Research workers' requirements for information vary, according to the stage that their work has reached. Sometimes one needs factual information, such as is assembled in reference handbooks. At other times, in contrast, one is nagged by an ill-defined need to find stimulation either from literature or from colleagues. There is a continuous spectrum of requirements between these two. The present work is concerned with needs that have an element of ill-definition, and that, perhaps, includes any that are not at the "factual" extreme of the spectrum. We felt that it was important to try to come to grips with the problem of serving a library user who is not able to formulate a precise query, and yet will recognize what he has been looking for when he sees it. A man, left to his own devices among the bookshelves, accomplishes searches of this

sort by browsing. Lancaster(1968) describes a type of search which undoubtedly occurs frequently in libraries:

> "Personal searches tend to be browsing searches. . . . Having found some promising references, [the seeker] locates the documents cited and, from the text and bibliographies of these, may be led to other sources or made aware of additional subject labels that might usefully be consulted in the tools with which he began the search. During this whole process, the 'information need' tends to be modified, to a greater or lesser extent, by what is found during the search, and the final set of documents, accepted by the searcher as 'useful' in relation to his requirements, may be somewhat different in character from the 'kinds' of documents he visualized as useful when the search commenced." - p181.

It seems that the notion of information in this context is extremely complicated. The concept of information has been discussed by Belkin(1974) and Brookes(1974), and they require that a suitable definition should take account of the state of the recipient's knowledge. It is because the information obtained (somehow) from a document alters the mental state of the reader, that he can conduct the type of browse described above. For the same reason, the "information content" of a book is very likely to differ from one reader to another. For the time being, therefore, it would seem that we need to read books and other documents to obtain certain types of information; and that fact retrieval from some kind of information machine is not sufficient. In designing a mechanical aid to literature searching, we should take the view expressed by the eminent chemist, Lord Todd(1967):

> "We must surely make the maximum use of computers and associated automation, but if we carry it to the point where the scientist no longer browses

> in the literature without first of all formulating questions then I beleive we shall do harm to science." - p9.

For some requirements - and they are not uncommon - the ideal search strategy would appear to consist of a visit to the shelves, and a perusal of the books themselves. The difficulty, of course, is in determining an arrangement of the books which assists the user. The arrangement should bring together literature on similar topics but, for the purposes of browsing, it need not take account of the fine detail in the subject matter. Hierarchical classifications, such as that of Dewey and the Universal Decimal Classification (UDC), are frequently used by libraries to generate a shelf order for the material. However, searching in the shelves is generally regarded as myopic, except in the smallest libraries, even though it is very often effective. In a large library, books which are potentially useful to one reader may be widely separated spacially, and the separation of the short, but very important, documents published in the many periodicals devoted to any particular subject is much more pronounced. Hence the need for reference retrieval.

The crucial characteristic of a reference retrieval device is that it aims to help the user to make choices from among unseen documents. The searcher wants a document for the (subjective) information it contains, so we have the very difficult problem of finding a proxy for the information, which must be very much smaller and more manipulable than the document itself. We need a symbolic description of the document - there

is no question of it being regarded as an alternative form of the information contained in the document, in the sense of information that we have in mind here. The most that we should aim for, at present, is a substitute which the user will interpret as meaning "this document may contain information I want". This is what class numbers (Dewey, UDC, Library of Congress, for example) and sets of keywords do for a document.

With a good descriptor language, documents which are relevant to a searcher's problem will have descriptions which he recognizes as being promising. The emphasis is on recognition: we are not saying that a query can be formulated in advance by the searcher to match those same descriptions. It seems reasonable to assume that there will be some similarity between the descriptions of documents which are relevant to the same query. But the nature of the similarity may be very subtle and hard to recognize by anybody other than the enquirer. In any case, conventional query formulation attempts to predict the descriptions of the required documents. These are the considerations that led the present author to the design of a reference retrieval system which offers no facilities for query formulation, in the usual sense, and proceeds on the basis of the user's reaction to references and document descriptions which it shows him.

There is another important dimension to the program design: it is the concept of dialogue used. Frequently, an enquirer can satisfy his information needs by talking to somebody with knowledge of an

appropriate subject. Telling him the broad area in which the problem to be solved lies, is relatively easy. Their dialogue (in which, by definition, each participates) refines the region of enquiry, until the subject expert understands the other's problem in his own terms. He may then be able to offer information which may lead to a solution. The dialogue is not always a simple question-and-answer interchange. The subject expert may miss the point and give a solution to the wrong problem; then the enquirer must bring him back on course - he must learn through conversation in what terms he should communicate his need. This is the approach adopted for our reference retrieval program. A computer program necessarily has a very limited view of the world; that is, the "terms" in which it can represent the user's problem area are rather primitive. This program's "knowledge" base is a richly connected network of references, subject terms and authors' names. It forms a model of the searcher's interest, derived from the network and continuously modified in the light of his reactions to references, which have been chosen for display according to the state of the model.

The program, named Thomas, was written for the IBM 360/67 at the University of Newcastle upon Tyne, and designed to communicate with a user at an IBM 2260 CRT character display terminal. The bibliographic data was obtained from the Medusa project in the Computing Laboratory of the University, reorganized into the network structure and accessed by the program from disk

storage. The literature covered is in the fields of medicine and biochemistry, and records originated at the US National Library of Medicine as Medlars (Medical Literature Analysis and Retrieval System) records. The indexing vocabulary in Medlars is strictly controlled, and each subject term in our network either belongs to that vocabulary, or is a synonym added to the Medusa system by the Newcastle team.

We now give a sample dialogue conducted by a medical research worker - an anaesthetist. This searcher was of the opinion that very few articles had been written on his precise topic. However, we had ensured that the test file contained references in his broader field of interest.

We shall indicate the lines supplied by the searcher by preceding them with the symbol ▶. This "start" symbol is used on the terminal to tell the user that he is required to type his next input, but it does not remain on the screen. A slight departure from the genuine computer displays is made in the interests of legibility in this printed form: we use the lower case alphabet here, whereas the IBM 2260 terminals are without those characters.

```
THOMAS, THE REFERENCE RETRIEVAL PROGRAM

Help can be obtained whenever the program has
displayed the start symbol by typing '?'
immediately after it.

Please give a short name for the search:
▶Alv.Resp.

Start searching:
▶pulmonary alveoli
```

The user has named the search, so that printed output will be identifiable. He has then typed the term for a subject related to his need. The program's model of the user's interest is centred on the subject keyword 'pulmonary alveoli', and includes a few references, one of which is (carefully) chosen for immediate display:

```
Influence of fasting on blood gas tension, pH,
and related values in dogs.; Pickrell et al, Am
J Vet Res,34,805-8,Jun 73
T. J.A.Pickrell, 2. J.L.Mauderly, 3. B.A.
Muggenburg, 4. U.C.Luft, 5. animal experiments,
6. animal feed, 7. arteries, 8. blood, 9. body
temperature, 10. carbon dioxide, 11. dogs, 12.
fasting, 13. hemoglobin, 14. hydrogen-ion
concentration, 15. irrigation, 16. lung, 17.
oxygen, 18. pulmonary alveoli, 19. respiration,
20. time factors
▶?
```

The searcher's request for assistance is answered by a display suited to this particular part of the dialogue:

```
There can be three parts to your statement (all
optional):
  1.  Your reaction to the reference just shown
(if any).  This must come first:
                  "Yes"  or  "No"
  2.  A selection from the names (authors) or
terms shown, by number.  A "not" in the statement
signifies rejection of all numbers that follow
it.
  3.  New names or terms (terms preferably in
quotes).
      The elements of the statement should be
separated by commas.
Examples:   'posture', 'circulatory system'
            Yes, not 11,12
            No, 7,13,4
            'heart rate'
            Yes
Press enter key when you are ready to proceed▶
```

When he has finished with the instructions, the user returns to the dialogue:

```
  . . .      10. carbon dioxide, 11. dogs, 12.
fasting, 13. hemoglobin, 14. hydrogen-ion
concentration, 15. irrigation, 16. lung, 17.
oxygen, 18. pulmonary alveoli, 19. respiration,
20. time factors
▶?
▶No, 10,17,19,20
```

The reference is not what he wants, although it has interesting aspects - carbon dioxide, oxygen, respiration and time factors. The model changes, and a new reference is displayed:

The arterial-alveolar nitrous oxide difference: a measure of ventilation-perfusion mismatch during anesthesia.; Markello et al, Anesth Analg (Cleve),52,347-50,May-Jun 73
1. R.Markello, 2. R.Schuder, 3. D.Schrader, 4. E. White, 5. aged, 6. anesthesia, inhalation, 7. blood, 8. carbon dioxide, 9. cardiac output, 10. human, 11. hydrogen-ion concentration, 12. methods, 13. middle age, 14. nitrous oxide, 15. oxygen, 16. partial pressure, 17. pulmonary alveoli, 18. respiration, 19. surgery, operative, 20. time factors, 21. ventilation-perfusion ratio
▶Yes

The enquirer was pleased to see this reference. It appeared to be an authoritative account of a different school of thinking on his own problem. The effect of his response is to enrich the model, drawing on the "knowledge" base, in the region of this document.

A Positive End-Expiratory Pressure - Nasal-Assist Device (PEEP-NAD) for treatment of respiratory distress syndrome.; Tummons, Anesthesiology,38, 592-5,Jun 73
1. J.L.Tummons, 2. blood, 3. carbon dioxide, 4. human, 5. hydrogen-ion concentration, 6. infant, newborn, 7. masks, 8. methods, 9. nose, 10. oxygen, 11. oxygen inhalation therapy, 12. positive-pressure respiration, 13. respiration, 14. respiratory distress syndrome
▶Yes, 13, not 6

He has mentioned 'respiration' for the second time now, but that makes no difference. Notice that the term that he first thought of - 'pulmonary alveoli' - is absent from this document's description.

Effects of continuous negative pressure on lung mechanics in idiopathic respiratory distress syndrome.; Baucalari et al, Pediatrics,57,485-93,Mar 73
1. E.Baucalari, 2. O.L.Garcia, 3. M.J.Jesse, 4. bicarbonates, 5. carbon dioxide, 6. esophagus, 7. female, 8. gestational age, 9. human, 10. hydrogen-ion concentration, 11. infant, newborn, 12. lung compliance, 13. male, 14. methods, 15. oxygen, 16. partial pressure, 17. pulmonary alveoli, 18. respiration, 19. respiration, artificial, 20. respiratory distress syndrome, 21. ventilation-perfusion ratio, 22. vital capacity
▶No, 19

This reference is judged not relevant, but the user makes it known that artificial respiration interests him. The term 'infant, newborn' (no.11) is present, implying that the reference has been chosen for display inspite of the fact that one of its descriptors has previously been explicitly rejected by the user. The assumption made about the user's intention when he says "not X" is that he does not want references to be selected on the basis of association with X, rather than that he does not wish to see anything to do with X.

Decrease in pulmonary capacity during lipid infusion in healthy men.; Sundstrom et al, J Appl Physiol,34,816-20,Jun 73
1. G.Sundstrom, 2. C.W.Zauner, 3. M.Arborelius, 4. adult, 5. carbon monoxide, 6. dietary fats, 7. human, 8. hyperlipemia, 9. male, 10. middle age, 11. oils, 12. oxygen, 13. parental feeding, 14. pulmonary alveoli, 15. pulmonary diffusing capacity, 16. respiration, 17. soy beans, 18. triglycerides, 19. ventilation-perfusion ratio
▶ [no reaction - user enters a null line]

The searcher prefers not to commit himself to a judgement on this reference. It is interesting, though not really pertinent to his present requirement. The model is not affected very much by this type of response: the user is saying, in effect, "no comment, give me another".

Cardiovascular function after pulmonary surgery.; Wronne, Int Anesthesiol Clin,10,27-39,Winter 72
1. B.Wronne, 2. adult, 3. aged, 4. arrhythmia, 5. blood pressure, 6. blood volume, 7. bronchial neoplasms, 8. cardiac output, 9. cardiovascular system, 10. human, 11. lung, 12. middle age, 13. postoperative complications
▶No

Changes of venous admixture with inspired oxygen in hyaline membrane disease and foetal aspiration pneumonia.; Corbet et al, Aust Paediatr J,9,25-30,Feb 73
1. A.J.Corbet, 2. E.D.Burnard, 3. anoxemia, 4. fetal diseases, 5. human, 6. hyaline membrane disease, 7. infant, newborn, 8. oxygen, 9. pneumonia, aspiration, 10. pregnancy, 11. pulmonary alveoli, 12. pulmonary circulation, 13. respiration, 14. ventilation-perfusion ratio
▶No

The anti-atelectasis factor of the lung. I; Lachmann et al,Z Erkr Atmungsorgane,137,267-87, Feb 73
1. B.Lachmann, 2. K.Winsel, 3. H.Reutgen, 4. animal experiments, 5. carbon dioxide, 6. extracorporeal circulation, 7. human, 8. lung, 9. lung compliance, 10. mice, 11. microscopy, electron, scanning, 12. models, theoretical, 13. pulmonary alveoli, 14. pulmonary embolism, 15. pulmonary surfactant, 16. rats, 17. respiration, 18. respiration, artificial, 19. review, 20. surface tension, 21. vagotomy, 22. ventilation-perfusion ratio, 23. work of breathing
▶Yes, not 11

The dialogue continued until a further 15 references had been displayed, as the user was obviously enjoying it, but no more relevant ones were found. We shall not follow the search in detail through to the point at which the user felt that he had seen all that the program had to offer. Before leaving the example, let us jump forward a few steps in the dialogue. The situation is that the user has rejected several references in a row and the program, which measures its own performance in the task of extracting favourable reactions from the user, now makes an attempt to get back on course. It shows him again a reference that he has previously judged relevant:

```
We are not doing so well now.  You may already
have the important references.
Please reconsider this document:

A Positive End-Expiratory Pressure - Nasal-Assist
Device (PEEP-NAD) for treatment of respiratory
distress syndrome.;  Tummons, Anesthesiology,38,
592-5,Jun 73
1. J.L.Tummons, 2. blood,    . . .
          .
          .
          .
14. respiratory distress syndrome
▶ No
```

Now, this judgement is a complete reversal of the earlier one, so the program has not succeeded in its course correction. The next display is:

```
We are not making progress.
Please reconsider this document:

The arterial-alveolar nitrous oxide difference:
a measure of ventilation-perfusion mismatch
during anesthesia.; Markello et al, Anesth Analg
(Cleve),52,347-50,May-Jun 73
1. R.Markello, 2. R.Schuder,    . . .
          .
          .
          .
          18. respiration, 19. surgery, operative,
20. time factors, 21. ventilation-perfusion ratio
▶Yes, 1, not 19,20
```

This was still the most important reference seen. The user had noted that the term 'time factors' was attached to several of the references, and had a wide variety of meanings, so he now stated that he was no longer interested in it. The response enabled the program to display a few more new references on topics in anaesthetics.

Naturally there are many aspects of the program which are not illustrated in the dialogue above. Nevertheless, it should give the reader an impression of the simplicity of dialogues with program Thomas. It can be seen that the program is not suitable in itself for large-scale, exhaustive literature searches. Even for such requirements, however, it may be a useful tool for getting a search underway. Finding a few references will help the searcher to decide what he is looking for, and should also provide a lead-in to the literature through chains of citations.

A full description of the program commences in Chapter 3, and occupies three chapters. Firstly, there is an account and discussion of the design, which attempts to explain why the program is the way it is. This is followed, in Chapter 4, by a more formal description, or specification, of the important features of the data base and program. Methods of recognizing subject terms, titles or names requested by the searcher, and the way in which the data base is organized in storage are matters that have received no mention so far in this introduction to the work. They are dealt with in Chapter 5. The process of recognizing user-supplied data is that of finding the record in the data base which best matches that data.

Chapter 6 is something of a digression. It was considered worthwhile to include an account of the methodology of design and programming used to implement Thomas. The principles of top-down, structured programming were applied to the construction of the software in a low-level language. The method was successful for experimental programming in an application field which does not fit conveniently within the scope of any established programming language.

In Chapter 7, we discuss the retrieval performance of the program and present the results of the trial searches. The evaluation of an on-line information retrieval system is difficult. One must decide whether to separate, for the purposes of measurement, the machine's contribution from the

user's. If we do not, then we are regarding the user as part of the system, and the evaluation must take into account his aims and performance. It has not been possible to observe a significant number of genuine searches, conducted by real users, within this project.

Before we embark on the material specific to our own work, however, we present a view of the subject of reference retrieval - the problems and some of the techniques used to tackle them. Chapter 2 is devoted to this.

Chapter 2

REFERENCE RETRIEVAL

1. The problem

Documentation, or "information science", is not a discipline in its own right, but rather a problem oriented field. Reference retrieval is one of the problems in its domain: how can an individual, with a desire to inform himself by reading, be aided in the selection of satisfying material from a large document collection? It may not be at all easy to recognize, objectively, a good solution (Kunz & Rittel,1972). From the enquirer's point of view, a retrieval device should not waste his time by presenting items that are not to the point, and it should not withhold items which would be influential in his current activities.

Several disciplines and technologies have been brought to bear upon the problem, either in attempts to understand it or to provide workable solutions: various branches of mathematics, including logic (e.g. Fairthorne 1961, Needham 1965, Hillman 1964, Bar-Hillel 1964); linguistics (reviews by Montgomery 1972 and Kay & Sparck Jones 1971, for example); psychology (e.g. Farradane 1967, Miller 1968, Treu 1971); engineering in various forms, including computer, communications and optical hardware (e.g. Overhage & Reintjes 1974), programming techniques and data organization (e.g. Salton 1968), and systems engineering (e.g. Vickery 1973, Kraft 1973). There are as many statements of the problem of reference retrieval, as approaches to the topic. We

shall now try to give an expression of the problem which is more precise (and manipulable) than that given above. It is based on the arguments concerning the classification of dynamic collections given by Fairthorne(1956 and 1958).

1.1 Ignorance and uncertainty

Fairthorne brought Boolean algebraic models of the retrieval process (from growing collections) into disrepute by pointing out that the principle of the "excluded middle" is violated in any realistic classification. In classical sentence logic, the principle of the excluded middle is that, for any proposition, p, $p \vee \sim p$ (i.e. p or not p) is a tautology. In other words, a proposition, such as "document d belongs to class A", is certainly either true or false. In reality, a document (or its reference) may have been marked in such a way that "the document belongs to class A" is known to be true, or it may have a mark which tells us that it does not belong to class A, or we may be in ignorance about its status as regards class A. When required to retrieve documents in class A, a system can find all known to belong to A and either include or exclude those about which it is really ignorant. If it includes them it is said (in Fairthorne's terminology) to be working in the all-but-not-only mode, otherwise it is in the only-but-not-all mode.

A system of logic, founded by Brouwer, has been developed (Heyting,1956) which rejects the principle of the excluded middle. Perhaps the most meaningful

name for the subject, from our point of view, is "constructivism". Fairthorne's and Hillman's reason (Hillman,1968) for wishing to weaken the logical model of retrieval in this way was that Boolean algebra "serves to prescribe decision operations only for those collections in which the complement of any set always exists and is, furthermore, describable."(Hillman,1968, p221). The need for description, or the specification of a construction, leads to problems when dealing with infinite sets, unless the excluded middle is rejected. We shall make limited use of these ideas; they just help us to discuss the problem of reference retrieval. It is difficult to view Brouwerian logic as a prescription for a system.

A document collection is not an infinite set, but the combination of documents and users as handled by an effective retrieval system cannot realistically be considered "closed". The ideal response to any particular query might be any of the subsets of the document collection, and normally the collection will be growing and the users changing. Figure 1 represents the situation at any particular time with respect to a class of documents, named A. C stands for the collection, and should not be thought of as necessarily static. The set K contains all documents which we know to belong to A, and the set N contains all documents which we know do not belong to A. Our (or the retrieval system's) knowledge is that which is derivable from the marks assigned to documents, by classifiers or indexers for example. Let us use the symbol $\bar{}$ to denote the

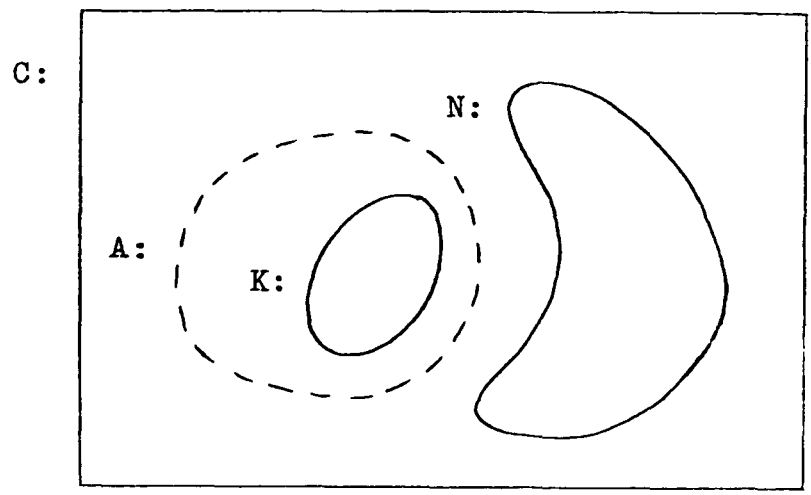


Figure 1.

ordinary Boolean complement; in this case

$$\bar{X} \overset{\text{def}}{=} C - X .$$

$\bar{X}$ is not necessarily constructible.

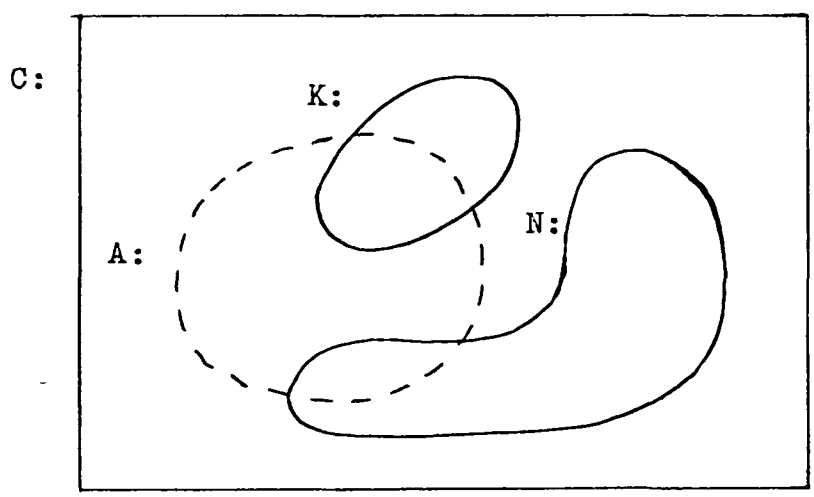
Our ignorance of the collection with respect to class A is $\overline{(K \cup N)}$. This set contains, for example, items that have not been classified, or have been classified wrongly, or have not been indexed sufficiently exhaustively for a decision to be made about their inclusion in A, the class required by the searcher. K and N are the only sets in the picture which are well defined, but they are artificial: what we wish to identify and retrieve is A. The problem of a retrieval system is to make either K or $\bar{N}$, or both, converge to A.

To go into the problem further, we need to know

something about the nature of the searcher's certainty that $K \subseteq A$ or that $N \cap A$ is empty. First of all we should dismiss the type of search (more likely to be done by a librarian than a library customer) which defines A to be that set of documents bearing some particular mark, say 611·34. The searcher describes K in the same way, and quite obviously A and K are co-extensive; there is no problem. It is quite another matter if the searcher defines A to be the set of documents dealing with what he understands by the word "intestines", say. If he accepts that the classifier or indexer attaches to the word "intestines" a meaning which is at least subsumed by his own understanding, then he can define K as the set of documents which have been assigned the index term INTESTINES, or the Dewey class mark 611·34, knowing that $K \subseteq A$ (having forfeited his right to deny it). He does <u>not</u> know whether $K = A$. If he accepts that all the Dewey numbers <u>beginning</u> with 611·34 are also used to classify documents dealing with the subject as he understands it, he can lay down a rule for constructing a larger $K \subseteq A$.

Now, although there must be some overlap in two individuals' understanding of words, for verbal communication between them to be possible, the assumptions we have made above are too strong to be plausible. As a result, we have only accounted for <u>ignorance</u> of the membership of some documents in the sought-after class, A. Factors such as lack of exhaustivity in indexing may cause some documents to be undetectable in a search for class A. If we make a weaker assumption about the

relationship between the meanings attached to a word by two individuals (the searcher, and the classifier interpreting the classification system), we can no longer assume that the searcher knows that his formulation produces a K that is entirely contained in A. We then have <u>uncertainty</u> that members of K are also in A; and, similarly, that members of N are not in A. The picture now looks like figure 2. K and N are still the



<u>Figure</u> 2.

well defined sets and are therefore still disjoint. However, if we retrieve K we no longer get "only but not all", and if we retrieve $\bar{N}$ we no longer get "all but not only". Fundamentally, however, the problem is still the same: attempt to bring either K or $\bar{N}$ into coincidence with A.

We have been very vague about the class A. It has been defined as the set of documents being sought by a particular user, and it has been noted that it might be

any member of the power set of C, the document collection. The concept of relevance is clearly involved here, and the debate in the literature on that topic is by no means concluded (for recent contributions, see Kemp 1974, Wilson 1973, Weiler 1973, D.J.Foskett 1972). So, for the time being, we must remain vague about A: that is why Brouwerian logic was introduced into the present discussion. But we can say a little more about it. The class A is a maximal set of documents, all of which the searcher will consider pertinent. It is not necessarily unique - the composition of the set may depend upon the order in which the searcher is presented with the references. It is maximal in that an enquirer will stop searching when his need for information is satisfied. Both of these aspects are related to the knowledge of the searcher at the beginning of the retrieval process, and the changes it undergoes during the search. Attempts to formalize the relationship between information and knowledge are being made by, for instance, Brookes(1974) and Belkin(1974).

2. Towards solutions

We have expressed the problem in terms of the necessity of specifying either a set K of documents "known" to be contained in A, the set which the searcher is after, or a set $\overline{N}$ "known" to contain A. To introduce the confusion that exists in real reference retrieval systems, we have pointed out that there is some degree of uncertainty in our knowledge that $K \subseteq A$ or $A \subseteq \overline{N}$. In practice, if we insist on a high degree of certainty,

then K is usually very small and $\bar{N}$ is very large.

The two most widely used measures of retrieval performance are precision, the proportion of retrieved references that are relevant, and recall, the proportion of relevant references in the collection that are retrieved. If the searcher uses a fairly certain definition for K, he may miss alot (low recall), but he will find little that is not relevant, i.e. he should get high precision. If, on the other hand, he is prepared to use a K which is less certain, he may be able to reduce his "ignorance" and thus obtain higher recall, but the uncertainty tends to reduce precision. Thus, there is a tendency for recall and precision to be inversely related, though this statement should be treated with caution (Cleverdon, 1972). We have discussed the isolated search. A system's performance is peculiar to the search and depends upon the way the system's features relate to the particular A sought.

The important features of a reference retrieval system, in the context of the present discussion are

(i) the indexing language, which places ultimate limits on the definitions that can be given for the set, K, and

(ii) the searching facilities, which determine how much of the potential power of the indexing language is usable.

2.1 Indexing

A detailed discussion of indexing is not within the

scope of this thesis. The topic is given extensive coverage in A.C.Foskett(1971), Lancaster(1972) and Vickery(1973). Lancaster(1968) describes "subject indexing as a two-step operation:

1. Deciding what a document is about (i.e. its subject matter);
2. Translating this conceptual analysis into index terms which act as shorthand symbols, or labels, for the subject matter of the document." - p3.

He points out that the interests of the intended users should influence the indexing. The symbols are taken, traditionally, from the vocabulary of an indexing language, which often also makes explicit a set of relationships between the symbols. Most British academic libraries use a "decimal classification" (e.g. UDC - British Standards Institution,1963), in which the vocabulary is strictly controlled and the relationships are implicit in the numerical symbols used. Further digits are added to a symbol for lower levels in the hierarchy. Other indexing schemes use words and phrases which occur in the natural discourse concerning the subjects represented in the collection. The vocabulary may be controlled by the use of a thesaurus, which will also give relationships between entries, such as "broader term", "narrower term" and cross-references. Some vocabularies are virtually uncontrolled: terms are taken from the titles, abstracts and even texts of documents in the collection. It is not easy to set up relationships between terms in such systems. K.P.Jones(1971) gives an interesting discussion of relationships in thesauri.

Another dichotomy in indexing techniques is the one

between the "pre-coordinate" and "post-coordinate" types - the dividing line is not very clear. In a system employing pre-coordination, each document is indexed by few terms, standing for complex concepts. To retrieve a document, a search formulation must specify the terms for component concepts in recognizably the same combination as was used to index it. In a post--coordinate system, more terms for simpler concepts are posted to each document and various combinations of them are coordinated at retrieval time, thus giving the searcher more versatility at the cost of greater scope for ambiguity ("false coordination" in the jargon of indexing). We have skated over the very involved topic of classification and indexing, giving brief, uncritical attention in very general terms to some of the major themes. A substantial experiment to evaluate the various commonly used methods relative to each other was done by Cleverdon et al(1966), and another, more recently, by Keen(1973).

Using the picture of the retrieval problem given in figure 2, we can now point out what various possible attributes of an indexing method can do to performance, i.e. to increase either the recall or the precision ratio.

Firstly, recall devices. These reduce the level of ignorance in the system. For any particular search, they allow us to specify a larger set, K, of documents which we can expect to lie within the required class, A, with some degree of certainty.

(i) Exhaustive indexing (discussed recently from the

statistical point of view by Sparck Jones,1973b). Terms for all topics covered in the document should be included in its description: the indexer does not know for certain what aspect of a document the searcher will find important.

(ii) Richly connected thesaurus. If, in determining K, we are to be able to infer from a search prescription, that a document which is not indexed with terms appearing in the prescription is, nevertheless, in A, then we shall need connections between terms in a thesaurus.

(iii) Specific indexing. Indexers are usually instructed to use the most specific term available to describe a topic (e.g. MEDLARS, see Lancaster,1969). This allows inferences based on class inclusion to be made.

Now we move on to <u>precision devices</u>. Uncertainty in the definition of K should be reduced by these.

(i) Choice of symbols. Vocabulary should be well accepted by practitioners in the subject field (Lancaster,1972, pp27-37).

(ii) Qualification of various uses of a word, so that meanings are not confounded.

(iii) Specific indexing (see iii, above). Needed because specific terms cannot be deduced from broader ones.

(iv) Term weighting (Maron & Kuhns 1960, Sparck Jones 1973, Salton & Yang 1973, Robertson 1974). Numerical weights associated with the terms assigned to a document can tell us which are the important topics covered or which terms are more

discriminating in the collection as a whole. (The zoologist who is interested in rats per se will not wish to encounter every experiment that has used rats. A system which enabled him to attach high weighting to the term RAT would give him better precision).

(v) Pre-coordination. This involves the indexer in specifying the relationships between concepts as expressed in the document. False coordination during search is reduced. Flexibility at the search stage is the main problem. Some sort of formal syntax must be used (e.g. Farradane et al 1973, Austin 1974, Coates 1973).

In comparison with the above, it is interesting to review Lancaster's list of "principle causes of search failure in information retrieval systems" (Lancaster & Fayan,1973, p141). His categorization is based on detailed analysis of failures during the MEDLARS evaluation (Lancaster,1969). Slight changes in terminology have been made for convenience.

	Recall failures	Precision failures
Index language	Lack of specific terms	Lack of specific terms
	Inadequate thesaurus structure	Defects in hierarchy
	Pre-coordination causing "over-preciseness"	False coordinations
		Incorrect pre-coordination
Indexing	Lack of specificity	Exhaustive indexing, causing retrieval

	Lack of exhaustivity	on peripheral topics
	Omission of important concepts	Use of inappropriate terms
	Use of inappropriate terms	
Searching	Failure to cover all reasonable approaches to retrieval	Strategy not sufficiently exhaustive
	Strategy too exhaustive	Strategy not sufficiently specific
	Strategy too specific	Use of inappropriate terms
		Defects in search logic
User/System interface	Request more specific than actual information need	Request more general than actual information need

The failures listed beside "searching" and "user/system interface" describe the ways in which a user can go wrong in defining K (or $\bar{K}$, if he is searching by rejection).

2.2 Searching

Indexing or classification - the process of characterizing documents for reference retrieval - is the crucial operation in a bibliographic information

system. The preceding account gives some of the general notions and, because search techniques are so dependent on indexing, this section will quite frequently digress into the topic of indexing. A search strategy takes advantage of the available document descriptions with the object of satisfying the need that prompted the user to search the literature. The strategy used will depend on the type of need and the amount of effort available for the search as well as the theoretical possibilities afforded by the indexing.

When all searching was done manually, it was generally considered that users of libraries would be served best by a hierarchically classified collection. By choosing, at each level of the hierarchy, the class that best matches the field of interest, the searcher can home in on a small set of potentially useful documents without even considering most topics covered by the collection. However, no hierarchical classification can suit all searches, and there will be occasions when it is necessary to extend the search across many branches of the tree. An interesting discussion on the nature of classification for retrieval is given by Sparck Jones(1970).

Post-coordinate indexing is an attempt at document description without an a priori hierarchy of classes. In its simplest form, each document is assigned a set of keywords, and a search formulation must specify which combinations of keywords an acceptable document should have. The so called "Boolean search" formulation is, perhaps, the most frequently used. Terms are

combined by logical connectives; for example

BIBLIOMETRICS or (STATISTICS and DOCUMENTATION)

would be used to select references which had been indexed either with the term BIBLIOMETRICS or with both STATISTICS and DOCUMENTATION. Another commonly used type of strategy is known as the "quorum search". The searcher specifies a list of terms and says how many of them must be present in the description of a document for it to be retrieved. One might, for example, require any two (or more) of the following four terms:

RELEVANCE, PERTINENCE, SUBJECTIVE, SIGNIFICANCE.

This is a special case of the technique of linear associative retrieval, in which a measure of similarity between possibly weighted query terms and document descriptions is used to rank documents by "closeness" to the query. Performing these types of coordination by hand is laborious and such methods did not become widespread until the advent of machinery to aid the task. Among the earliest mechanical systems were optical coincidence cards (Batten,1947), and edge--notched cards (Mooers,1951). The former is an inverted file - a card for each subject term - and many computer-based systems employ the same principle in their file organization (Lefkovitz,1969). Mooers' system is a mechanical version of content-addressable memory. Linguistic problems are more serious in post--coordinate indexing: an example, false coordination, has already been mentioned.

Pre-coordinate systems are linguistically more satisfactory for the human searcher, because the syntax

in the description makes the relationships between the component concepts clear. The linguistic subtleties make automatic searching difficult, however.

2.2.1 Automation of delegated searching

Now, having given the general picture, we shall concentrate on aspects of the automation of reference retrieval. Very nearly every text on information retrieval begins by pointing to the "information explosion" as an urgent reason to enlist the aid of fast machinery. They are probably right. Both the literature and the user population are growing, so the total volume of indexing increases, and so should its complexity. Searches also become ever more arduous as more discrimination is needed. If we are to delegate a substantial portion of the work to a machine, we must either give the machine linguistic skills (particularly in the area of semantics), or we must find efficient ways of dividing the tasks between man and machine (Doyle,1965). The questions to be answered are: how should the user express his need? having answered that, how should the collection be described? then, what search strategies and matching algorithms should be applied?

The answer most frequently given to the first question is "in whatever way seems natural to him". Moyne(1969) gives reasons for using a natural language to express queries. Apart from ease of use by casual users, he points out that "natural languages are highly economical and efficient systems" for communication of

complex messages. There is nothing new here: specialized information services have received queries in natural prose for a long time. An information worker constructs a formal query, using all his knowledge of the document collection and its descriptive adjuncts - this is called "delegated searching". In fact, he will analyse the query in much the same way that the documents have been analysed on entry to the system.

Automatic systems exist which emulate this type of service. Abstracts or full text of documents are prepared for machine reading, and analysed for content indicators; requests are treated in the same way, and the resulting representation compared against the document descriptions. The most exhaustively documented system of this type is a versatile collection of experimental modules called the SMART system (Salton,1971). Numerous comparisons have been made between system performances observed with various linguistic algorithms, ranging from simple word stem extraction, through the use of thesauri to normalize vocabulary, to the construction of parse trees for phrases. Retrieval is usually performed in SMART by ranking the whole collection (100 - 1000 documents) according to their similarity to the request; documents within a certain distance of the top of the list are considered retrieved. The more complex syntactic representations which were prominent in earlier papers (Salton 1962, Salton & Sussenguth 1964, Salton 1966) have produced disappointing results: "when the phrase generation procedures using simplified syntax are compared with other, simpler, content analysis

methods which include no structural or semantic components, the surprising conclusion is that on the average better results are obtainable without the syntactic components than with them." (Salton,1973, pp259-60). Montgomery(1972) is highly critical of the syntax analysis procedures used in SMART, however, so Salton's conclusion may not be so surprising. As for the more straightforward processes, which reduce documents and query to weighted term vectors, Salton (1972) shows that they give results comparable to those obtained by conventional human indexing and Boolean searching (with a collection of 450 documents).

Another system which handles natural language (documents and queries) is BROWSER (Williams,1969). Significant terms are extracted from the text using a dictionary of "root words". Dictionary entries have "information values" attached to them which vary inversely as the total number of occurrences of the root word in the document corpus - they are indicative of the usefulness of the term in searching. Sparck Jones(1972b) defines "term specificity" in a very similar way (i.e. as a statistic associated with a term's usage in a set of document descriptions).

A rather more complex linguistic analysis is performed by the LEADERMART system (Hillman,1973 and 1968). Sentences are decomposed into logical relations between noun phrases. The noun phrases, it is presumed, are what the sentence (and its containing document or query) is about, and the relations involved determine a weighting for the noun phrases, as well as providing

information for partitioning the collection (i.e. classifying it).

The descriptions, above, of the three systems - SMART, BROWSER and LEADERMART - are, of course, incomplete; we have concentrated on what they do to their natural language input. Their common feature is that they process requests in the same way as the document texts in their files, which is the answer to our second question - how should the collection be described? - if we assume that the user should indeed express his need (to an automatic system) in his natural language. So the indexer has disappeared from the scene, and the author is communicating directly with the potential reader. Now that each is using his own language (with no interposed, controlled indexing language), the third question - what search strategies and matching algorithms should be applied? - has no simple answer. We need to know precisely what are the connections between the words (symbols) we use and the concepts we are trying to communicate, and that is the province of semantics.

2.2.2 Semantics

The discussion, here, of semantics will be very brief: there are many review articles which cover the subject (Kuno 1966, Bobrow et al 1967a, Montgomery 1969, Kay & Sparck Jones 1971, Pacak & Pratt 1971, Montgomery 1972). All of these reviewers are interested in making linguistics work in the development of man-machine communication, and all lament the lack of guidance from

theoretical studies, particularly studies of semantics. Among theoretical linguists, semantics has received comparatively little attention, and every prominent semanticist has his own theory. One significant common thread that runs through all the work in this field is that an important aspect of the meaning of words is the relationships they contract with each other. Whether the relationships determine the meanings (Lyons,1968), or vice versa (Katz & Fodor,1963) is a matter for debate, as is the question of the nature of the relationships; whether they can be classified into types - e.g. synonymy, antonymy, inclusion - (Sparck Jones,1965).

On the practical plane, it has been shown that a certain amount of "understanding" can be displayed by programs which manipulate networks of words (Quillian 1968, Simmons et al 1968, Simmons & Slocum 1972). However, although it is clear that the environment of a word in a simple (though large) network can be highly suggestive of its meaning to a human observer (Doyle 1961, and see figure 3 for an illustration), much more is needed to tell him (or a machine) how to use the word. The success of Winograd's program SHRDLU (Winograd,1972), supports the intuitively obvious hypothesis that the understanding of natural languages (i.e. that which brings forth an appropriate response to a message conveyed in a natural language) demands knowledge of the area of discourse, which includes the discourse itself, and the ability to solve problems in that area. The meanings of words are embodied in

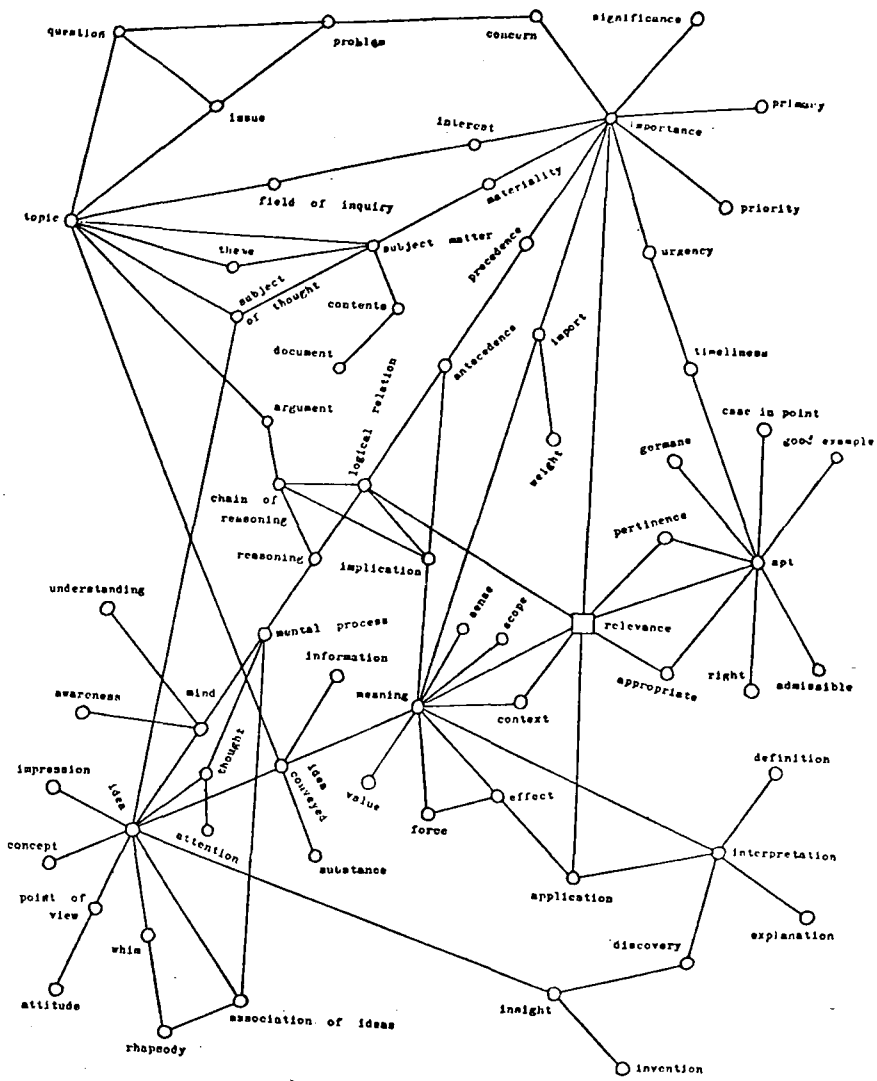


Figure 3. The verbal environment of "relevance". All the associations drawn in this picture were taken from Roget's Thesaurus (Penguin edition). The reader may judge how much of the meaning of the words is evident from the figure. The present author is reminded, by it, of much of the substance of recent published discussions on relevance in the information retrieval context.

procedures, which may invoke manipulation of the program's model of the world. If it is true that proper use of natural languages cannot be divorced from other mental activity, and knowledge, then we must make do with much less in our mechanical intermediaries between author and reader. The amount of knowledge handled by a useful information service is vast.

So, although relatively simple syntactic analysis of document texts may produce acceptable symbolic characterizations (by conventional standards), one should not yet expect enormous benefits from using natural language as a medium for expressing a search request. Successful operational systems which use this mode of communication (BROWSER and LEADERMART, for instance) probably depend for success more on interaction with the user, on-line, than on their ability to make something of his English. We shall come back to the question of interaction in a later section, but first we consider some of the uses to which relationships between words have been put, in attempts to enhance reference retrieval performance.

2.3 Associations and clusters

A great deal of work has been done on the discovery and use of associations between words, and other entities involved in reference retrieval. The background to this activity, linguistic, psychological and philosophical, has been discussed by P.E.Jones(1965), and Tague(1970) has written a useful review.

Associations occur in various ways:

(i) "Semantic" relations between words. Hierarchies and cross-references in subject catalogues and thesauri for information retrieval (K.P.Jones 1971, Sparck Jones 1972a). These are the plausible relations: we tend to think of them as inevitable, derivable from the nature of the world. This is probably largely illusory, as indicated by the fact that classifications become out of date and vary from one library to another.

(ii) Statistical relations between words. This is an association with a measure instead of a type. Words are meaningfully associated if they tend to co-occur (Doyle 1961, Maron & Kuhns 1960 are prime examples among many who assert this). If the tendency is strong enough, the words can be regarded as synonyms for retrieval purposes because, used as index terms, they are nearly interchangeable - this is the justification for the keyword classification procedures used by Sparck Jones(1971). Suppose, now, that we find the words which tend to co-occur with the statistical associates of a particular word. These are what Stiles(1961) called "second generation terms", and are the words which tend to occur in the same context as the original word. Some of them will be synonyms of that word, in the linguist's sense (Sparck Jones,1965). The ideas of semantic and statistical second generation links were brought

together by Gotlieb & Kumar(1968) when they analysed the statistical association of pairs of terms in the Library of Congress subject headings, using the existing hierarchy and cross-references without distinguishing between the types of relationships. A large scale statistical term association experiment was done by Jacquesson & Schieber (1973) using a file of 40,000 references, indexed by 1400 terms. They found that even in their strictly controlled indexing vocabulary (i.e. where there should have been no synonyms), there was, in fact, an appreciable amount of overlap in the use of words.

(iii) Similarity relations between documents. The "distance" between documents can be worked out by considering the extent to which they are similarly indexed (Jardine & van Rijsbergen 1971, Rettemeyer 1972, van Rijsbergen & Sparck Jones 1973).

(iv) Bibliographic coupling. Assuming that authors tend to cite papers which have some bearing on their subject matter, another meaningful distance measure between documents is obtainable from their bibliographies (Weinberg 1974, Zunde 1971, for example). Gray & Harley(1971) bring together these two concepts of document similarity (iii and iv). They use bibliographic coupling to suggest terms to the indexer.

(v) Arbitrary user-specified association. By this, we mean links between records created by a user, as envisaged by Bush(1945). He laid down design

> principles for a personal filing mechanism in which any document, note, correspondence and so on would be stored and linked to existing records in whatever way its user wished. Searching would be done by following trails of associations. Several systems have been constructed along these lines (Glantz 1970, Treu 1970, Robinson & Yates 1973, Engelbart et al 1973). The facility for adding arbitrary links to a communal information structure, preferably under some sort of control, might be a useful addition to a document retrieval system, but we shall not discuss it further here. Reference retrieval is concerned with bringing to the notice of the user previously unknown documents; not with organizing the information for him after he has become aware of it.

We now turn to uses to which associations have been put in reference retrieval. Two objectives have been sought; they use similar techniques and are inter-dependent, but should be distinguished. Co-occurrence figures have been used to generate classes both of documents and of index terms. The main motivation for the former is to achieve efficiency of file searching by cutting down the amount of the document file which must be examined (this is very important in systems such as SMART which retrieve by measuring the association between documents and query, and ranking the documents). The motivation for grouping index terms is to enable the system to expand a query (mainly) to achieve higher recall. As Stiles(1961) put it:

> "Literally hundreds of terms may have been used to index documents on the various aspects of a particular subject and yet we must grope for just the right set of terms." - p271.

The main stream of automatic classification (or clustering) methods (whether of documents or index terms) can be summarized as follows:*

The documents in the collection are assumed to be described by lists of weighted index terms. In other words, each document is represented by a vector whose dimension is equal to the number, t, of terms in the vocabulary and which consists of the weights of all the terms, as applied to the document. If a term is not applied (posted) to a document, its weight is zero in that document's vector. Frequently, in practice, the only weights used are 0 and 1. The whole collection of d documents is then represented by the $d \times t$ matrix, M, having as its rows the d document vectors. Now, a matrix product operator, $\otimes$, is defined and applied to M and its transpose, M^T, to form a similarity matrix:

either $S_d = M \otimes M^T$, for document clustering,

or $S_t = M^T \otimes M$, for index term clustering.

The result is a square, symmetric matrix giving a measure of the similarity between every pair of documents (S_d) or terms (S_t). The operator $\otimes$ is usually defined for matrix operands A ($p \times q$) and B ($q \times r$), to give a $p \times r$ matrix product $C = A \otimes B$,

---

*For document clustering, see Jardine & van Rijsbergen (1971, good review included), van Rijsbergen(1974), Salton(1971) Part IV Cluster generation and search, Rettemeyer(1972) and Crouch(1973). For index term, or keyword, classification, see Sparck Jones(1971), Needham(1965), Augustson & Minker(1970), Minker et al (1973), Gotlieb & Kumar(1968), Borko & Bernick(1963, 1964), Stiles(1961).

where $$C_{ij} = \frac{\sum_{1 \leq k \leq q} A_{ik} B_{kj}}{N_{ij}}, \quad 1 \leq i \leq p, \ 1 \leq j \leq r.$$

N_{ij} is a normalizing factor, a function of the vectors $\{A_{ik}\}$ and $\{B_{kj}\}$, $1 \leq k \leq q$. For example,

$$N_{ij} = \sum_{1 \leq k \leq q} (A_{ik} + B_{kj}).$$

Having obtained the similarity matrix (S_d or S_t), the associations can be found by deciding upon a threshold, θ, and replacing each element of the matrix by 1 if it is not less than θ, or 0 otherwise. The result is the adjacency matrix representation of an association graph. A simple example should clarify these generalities.

Suppose we have 5 documents indexed by 6 different terms, $t_1 - t_6$, without weights, as follows:

$$d_1 = \{t_3, t_5, t_6\}, \qquad d_2 = \{t_1, t_3\},$$
$$d_3 = \{t_1, t_2, t_3\}, \qquad d_4 = \{t_2, t_4, t_5\},$$
$$d_5 = \{t_4, t_5, t_6\}.$$

Then $$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

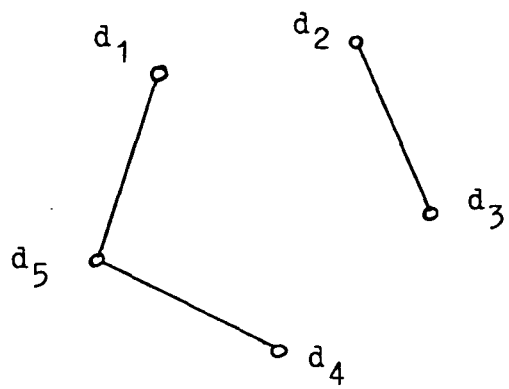
Using the particular definition of $\otimes$ given above, the document similarity matrix is

$$S_d = \mathbb{M} \otimes \mathbb{M}^T = \begin{pmatrix} 1/2 & 1/5 & 1/6 & 1/6 & 1/3 \\ & 1/2 & 2/5 & 0 & 0 \\ & & 1/2 & 1/6 & 0 \\ & \text{symmetrical} & & 1/2 & 1/3 \\ & & & & 1/2 \end{pmatrix}$$

Now we choose a threshold, $\theta = \frac{1}{3}$, say. Then the adjacency matrix is

$$A_d = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ & 1 & 1 & 0 & 0 \\ & & 1 & 0 & 0 \\ & \text{symmetrical} & & 1 & 1 \\ & & & & 1 \end{pmatrix}$$

which corresponds to the document association graph:



(we omit loops). We shall discuss similarity between documents again in Chapter 3.

Having established an association graph, between terms or between documents, there are various ways of forming classes, or clusters. Examples are maximal complete subgraphs (cliques, within which each node is connected to every other node), maximal connected subgraphs (every node in the subgraph is reachable from every other node), stars (one node is adjacent to every other node). Augustson & Minker(1970) and Sparck

Jones(1971) discuss the possibilities. Not all techniques produce disjoint classes and the threshold (θ), which obviously affects the association graph, also affects the clusters obtained. The need to select a somewhat arbitrary threshold led Needham(1965) to define a "clump", using the similarity (rather than the adjacency) matrix. An object is a member of a clump if the sum of its similarities with all the other members of the clump is greater than the sum of its similarities with all non-members. In contrast, Jardine & van Rijsbergen(1971) produce a hierarchy of document clusters by systematically varying the threshold.

The detailed results of applying these techniques are given in the literature already cited. Those who are investigating document clustering must show that improved efficiency is not accompanied by serious loss in retrieval performance. The most thorough evaluation of the many possibilities for query expansion by term classification is contained in Sparck Jones(1971). The conclusion seems to be that the best combination of clustering techniques tried performs significantly, but not substantially, better than simple term searching. There is some later work (Sparck Jones 1973a, van Rijsbergen & Sparck Jones 1973) which explains the performance of keyword classification in terms of characteristics of the document collection (derivable from a similarity matrix), with respect to the set of test queries.

2.4 Interaction

Throughout this chapter, so far, we have had a

particular type of search in mind. The enquirer has a need for information which is well formed in his mind, and he is able to express it quite precisely as a query. We have discussed the problems that arise when we assume that, in order to satisfy the searcher, the system must match, in some precise sense, the query with references in its store. Lancaster(1969) estimated (with qualifications) that the MEDLARS demand search service achieved, on average 58% recall and 50% precision. The construction of the formal search profile was delegated to trained search editors. Relevance was assessed by the end-users, and relatively low degrees of relevance were accepted for computing the above figures (i.e. "minor value" articles are considered relevant). If we accept that results obtained in small scale experiments (particularly the SMART document and query analysis trials, and the automatic classification tests of Sparck Jones) are valid for large collections, then the average recall and precision figures could increase by about 10%, i.e. to around 64% and 55% respectively. The studies which have been reviewed are attempts to find out how far we can go in creating machinery to which a man can delegate his search, and they are important as such.

However, now that facilities are widely available for the interactive use of computers, solutions to the difficult linguistic problems are not required so urgently. We have far more scope now for interleaving mechanical and intellectual work. In 1965, Doyle wrote that there were two alternative attitudes to the solution of linguistic problems:

> "(1) We can seek to make our procedures approach in complexity those used by the human intellect, and this appears to be the route preferred by most of the research people; or (2) we can try to take advantage of the fact that humans are experts in handling language, and have them work in senior partnership with computers." (Doyle,1965, p238).

Combine this with the fact that even among those with apparently well defined needs "a characteristic feature of this [information gathering] process is that the scientist's original inquiry or interest is invariably modified and restructured on the basis of the information presented to him." (Hillman,1968). The obvious result is that we should design the means whereby a searcher may explore the collection, gradually refining his request. When we reconsider the traditional distinction between browsing and searching, in this light, we find it so hazy that we are forced to abandon it (Herner,1970). This is not to say that all searches are alike: on one occasion, a scientist may want to look for a small amount of material to stimulate him, at another time he may wish to do a thorough literature search on some topic, and there are other possibilities.

Lancaster & Fayen(1973) have recently published a comprehensive state-of-the-art account of on-line information retrieval systems, in which they give brief desciptions of about 30 major operational systems, mostly of North American origin. That is clearly a small proportion of the systems now in existence. Most of the work currently being done is concerned with man-machine interface engineering (Walker 1971, Martin et al 1973) and this is outside the scope of this thesis. We are concerned with the information structures and processes

which can assist a user in his search. We shall not undertake an extensive survey of systems here: many differ from each other only in superficial detail (regrettably, some systems are incredibly verbose).

At the very least, an interactive retrieval system must help the user to find the appropriate words, and must provide facilities for developing his query, having shown him something from its files in response to his previous messages. Williams(1971) has given a much more detailed list of capabilities that he considers important for a browsing system.

One of the major variations between systems is in the indexing vocabulary used. Some systems (e.g. RECON - Wente 1971; BOLD - Burnaugh 1967; the Medusa system - Barber et al 1973) use a controlled indexing vocabulary, and incorporate appropriate devices for exploring it: on-line thesauri, with procedures for following the links between terms. By having related terms displayed the user is able to find words which he may not otherwise have thought of. The user may build up, in stages, Boolean or quorum search strategies. Another important component of these systems is a large "entry vocabulary", that is a set of words and phrases which are not in the restricted vocabulary of the indexers, but are commonly used by searchers. They are linked to the preferred terms. Higgins & Smith(1969) have suggested a way in which the entry vocabulary could be extended by the users.

Other systems search the free text of titles, abstracts or whole documents, having created an index to all occurrences of every significant word or stem.

Significance is usually determined by the word's absence from a "stop list" of common syntactic function words (articles,prepositions, etc.). Examples of this type of system are the Epilepsy Abstracts Retrieval System (Porter et al,1970) and the STATUS programs (Price et al,1974), which are used to search legal texts. One can often form Boolean queries in these systems, specifying that the combination should occur within a single sentence or larger unit of text. Another type of search is for a pair of words occurring within a certain distance of each other. Where there is no controlled indexing vocabulary, no thesaurus is likely to exist. It may be possible to display the neighbours of a word in an alphabetical list, but on the whole free text systems rely on further words being suggested to the user when he is shown relatively large pieces of text.

To illustrate the concepts involved in interactive retrieval, we give a brief desciption of, and a sample conversation with a particular operational reference retrieval program. The Medusa system, developed at the University of Newcastle upon Tyne is suitable for this purpose for three principle reasons: Firstly, it operates on a well-known data base - MEDLARS - with its controlled vocabulary of Medical Subject Headings (MeSH). Secondly, it has features which clearly show off the benefits of interactive search. Finally, the data and test queries used in the new work described in this thesis were obtained from Medusa files, as explained in Chapter 7, section 2.

2.4.1 An example: Medusa

The descriptive material and sample run in this section are adapted from the User Manual for Medusa, prepared by J.A.Hunter (University of Newcastle upon Tyne,1974). Medusa is an on-line reference retrieval system which runs on the IBM 360/67 computer at the University of Newcastle upon Tyne, using MEDLARS (Medical Literature Analysis and Retrieval System) data from the U.S. National Library of Medicine. Medusa is designed for direct use by medical research workers (Barber et al,1973).

Two means of accessing the system are provided; Current Awareness and Retrospective Medusa. Both systems allow the user to formulate an identical search, but differ in the manner of searching the data available.

CURRENT AWARENESS MEDUSA is intended for those users who want to keep up to date with the current literature; they will expect to return to the system each month, or at least every three months, and search the data acquired since they last used the system. The database kept for the Current Awareness system is the latest three months of the file. This is updated as new information arrives, the oldest month being dropped and the new months citations added. There are about 45,000 citations indexed from 2,200 journals for papers written in English, French and German. Users running current awareness searches may retain up to four different profiles from session to session as it is anticipated that they will wish to modify their search criteria as their work progresses.

RETROSPECTIVE MEDUSA is intended for users requiring a simple search on a particular topic from as large a database as possible. Some 110,000 citations are available for searching taken from 1,150 journals over the past year. The citations are restricted to those written in English. A Retrospective session is self-contained; that is, any search formulation is lost when the session ends. A special SAMPLE command, q.v., is supplied to permit checking of a search against the latest block of citations before it is used to access the whole database.

Medusa citations are indexed with terms selected from a thesaurus of 10,000 medical subject headings(MeSH). The user has to formulate his search using terms from this thesaurus. The main object of the Medusa system is to enable the user to find the correct terms for his subject. The task of finding all relevant terms is made

easier by their organization into categories - e.g. neoplasms, musculoskeletal system, vertebrates, surgery. The general term is at the "top" and the more specific terms appear below, down to four levels - e.g. vertebrates - mammals - rodents - mice. The terms above and below a particular term can be displayed easily on the terminal. Going "up", "down" and "across" the category structure is the way in which a user finds available terms. Papers are indexed under an average of ten main headings. An important point about the selection of terms is the use, by the indexers, of the most specific term for a subject. In addition to the 10,000 MeSH terms, there are 7,000 entry terms which, in most cases lead to synonymous MeSH terms. Some entry terms call up compound search expressions instead of single terms.

There is a repertoire of commands for exploring the thesaurus, constructing search prescriptions, and retrieving references. The user may introduce terms at any time; the system will assign to them short codes for easy reference later. We start with thesaurus exploration commands:

DOWN followed by a term code, will reveal the more specific terms, if any. If successful, this command will also generate the category term, identified by the C prefix. This refers to all of the terms in the relevant category below the original term.

UP will reveal the broader term.

ACROSS will reveal related terms at the same level.

XREF will reveal any cross references to different categories. These are indicated by an X printed in the display of a term.

Qualifiers are sub-headings which may be linked to main headings and categories to restrict the context in which they retrieve references.

QUAL followed by a code will print a list of those qualifiers which may be legally linked to a term when forming a search statement.

LIST followed by a character string of three or more letters will cause the system to print out dictionary entry names which start with those letters.

There are also commands to remind the searcher of details of previously used terms. This is particularly useful in Current Awareness Medusa where he may come back to a search profile after some time. (We omit their definitions here).

Now we come to the commands for formulating searches and performing them:

COMBINE followed by a term code, or by a group of codes separated by one of the operators AND, AND NOT, OR, LINK (for attaching qualifiers to terms), will form a search statement. The system will print out an R code number and give a rough estimate of the number of citations liable to be retrieved by its use. The R code number can be used in subsequent COMBINE commands and thus a complex profile can be built up, without the need to construct it in one error-prone step.

SEARCH followed by an R term causes the system to search for citations satisfying the criteria of the term. Citations found are printed out on the terminal in sufficient detail to enable a user to locate them, and with their associated index terms and sub-headings. An asterisk against an index term means that it is a "print" term, and appears against the reference to the citation in the indexing journal Index Medicus. If index terms are not desired, as with a profile of established reliability, they can be suppressed to give faster printing.

SAMPLE is available only in Retrospective Medusa. It is similar to SEARCH in use but only searches the most recent month of citations.

In the run of Retrospective Medusa which follows, all comments, i.e. lines not printed at the terminal, appear in lower case. Lines typed by the user are underlined.

MEDUSA INFORMATION RETRIEVAL SERVICE

PLEASE INDICATE WHETHER YOU WISH TO USE CURRENT AWARENESS OR RETROSPECTIVE MEDUSA BY TYPING "C" OR "R".
?R

RETROSPECTIVE MEDUSA

THE RETROSPECTIVE SERVICE IS AVAILABLE FROM 12.00 - 14.00 AND FROM 16.00 - 19.00 EACH WEEKDAY. SHOULD YOU SIGN ON DURING THESE PERIODS AND FIND THE SYSTEM NOT AVAILABLE, PLEASE RING MEDUSA STAFF ON 0632-28511 EXT. 2761.
SYSTEM HOLDS CITATIONS FOR APRIL AND MAY 1974, OCTOBER 1973 TO MARCH 1974, APRIL TO SEPTEMBER 1973

* SIGN INDICATES THAT SYSTEM IS READY FOR A REPLY

```
 SEARCH NUMBER   1 : USERCODE, NAME, TITLE ?
*0109, J.A.HUNTER, DEMONSTRATION SEARCH
 ENTER TERMS. START BY TYPING IN A MEDICAL TERM RELATED
 TO AN ASPECT OF YOUR SEARCH
*STUDENTS
 M1=STUDENTS                    T285       1962          X
          DN 6      A EDUCATION                    (ANTHROPOLOGY,
 EDUCATION,)

          DN 0      B EDUCATION, NONPROFESSION(ANTHROPOLOGY,
 EDUCATION,)

          DN 1.5    C NAMED GROUPS (NON MESH)

          DN 0      D OCCUPATIONS                  (SOCIOECONOMIC
 FACTORS   (POPULATION CHARACTERISTI))
```

The term "students" has been assigned the code M1. T285 gives the number of citations indexed under this heading, and 1962 gives the date of introduction of the term. X indicates the presence of one or more cross references. This term is in four categories A, B, C and D. In the C category, the code DN 1.5 means that there is one term in the next lower level of the category, and five in the level below that. In the A category, "education" is the broader term, and the information which follows it gives the category structure above it.

```
*DOWN M1A
 C1=STUDENTS                    TT858
 M2=STUDENTS, DENTAL            T93        1962              A,B,C
 M3=STUDENTS, HEALTH OCCUPAT    T21            NEW TERM      A,B,C
 M4=STUDENTS, MEDICAL           T267       1962              A,B,C
 M5=STUDENTS, NURSING           T174       1962              A,B,C
 M6=STUDENTS, PHARMACY          T9             NEW TERM      A,B,C
 M7=STUDENTS, PREMEDICAL        T9         1962              A,B,C
```

Here "down" has generated for the A category the category term C1, which encompasses all terms below and including "students" in that category. TT858 gives the total tally - the number of times the terms in the group have been used in indexing references.

```
*FULL M4
 M4=STUDENTS, MEDICAL           T267       1962
          DN 0      A STUDENTS                     (EDUCATION
      (ANTHROPOLOGY, EDUCATION,))

          DN 0      B STUDENTS, HEALTH OCCUPAT(STUDENTS
      (NAMED GROUPS (NON MESH) ))

          DN 0      C STUDENTS, HEALTH OCCUPAT(HEALTH MAN-
 POWER(FACILITIES MANPOWER SERV))

*XREF M1
 M8=STUDENT DROPOUTS            T48        1962          X A,B,C
*STUDENT HEALTH SERVICES
```

```
M9=STUDENT HEALTH SERVICES   T114      1962
          DN 0     A HEALTH SERVICES          (FACILITIES
MANPOWER SERV)

*COLLEGES
M10=UNIVERSITIES                T435      1962
          DN 0     A SCHOOLS                  (EDUCATION
     (ANTHROPOLOGY, EDUCATION,))
```

Note the action here in the case of a synonym being entered. "colleges" is held in the dictionary as a pointer to "universities", for which it generates a code.

```
*COMBINE M1 OR M8 OR M9 OR M10

R1= M1 OR M8 OR M9 OR M10
EXPECTED RETURN: LARGE
```

Here large means 25 or more citations would be retrieved.

```
*SAMPLE R1

FIRST CITATION FOUND IN  21 SECS

CIT NUM 00290919
HOWELL R    CROWN S     HOWELL RW
PERSONALITY AND PSYCHOSOCIAL INTERACTIONS IN AN UNDER-
GRADUATE SAMPLE.
BR J PSYCHIATRY VOL123  699-701  DEC 73
*PERSONALITY ASSESSMENT          *STUDENTS
 ADOLESCENCE                      ADULT
 AFFECTIVE DISTURBANCES           FACULTY
              (DIAGNOSIS)
 FEMALE                           HUMAN
 JUVENILE DELINQUENCY             MALE
 PSYCHOMETRICS                    SCHOOLS
 SEX FACTORS                      SOCIAL CLASS
```

In this print out of a reference, the terms marked with an asterisk are "print" terms which would appear in Index Medicus. Terms in brackets are qualifiers, e.g. (diagnosis).

```
CIT NUM 00290921
DUDDLE M
AN INCREASE OF ANOREXIA NERVOSA IN A UNIVERSITY
POPULATION.
BR J PSYCHIATRY VOL123  711-2  DEC 73
*ANOREXIA NERVOSA(OCCURRENCE)  *STUDENTS
 ADOLESCENCE                    ADULT
 EDUCATIONAL STATUS             ENGLAND
 FEMALE                         HUMAN
 INFANT NUTRITION               MALE
 OBESITY                        UNIVERSITIES
```

```
 CIT NUM 00291303
 CAMPBELL LP
 MODIFYING ATTITUDES OF UPPER ELEMENTARY STUDENTS TOWARD
 SMOKING.
 J SCH HEALTH VOL44  97-8  FEB 74
 *HEALTH EDUCATION                *SMOKING(PREV CONTRL)
  ADOLESCENCE                      ATTITUDE TO !!

 ATTENTION INTERRUPT
  MALE                             STUDENTS
```

Here the interrupt key has been used to stop a search. Note that printing of the current citation is completed before control is returned to the user.

```
*DRUG ADDICTION
 M11=DRUG ADDICTION              T816      1962              X
           DN 3     A DRUG ABUSE                  (PSYCHIATRY
      )

           DN 0     B SOCIOPATHIC PERSONALITY (PERSONALITY
 DISORDERS   (PSYCHIATRY          ))

*ACROSS M11A
 M12=GLUE SNIFFING               T18          NEW TERM   A,B,C
 M11=DRUG ADDICTION              T816      1962          X A,B
*UP M11A
 M13=DRUG ABUSE                  T708      1962          X A,B,C
*COMBINE R1 AND M13

 R2= R1 AND M13
 EXPECTED RETURN: SMALL
```

Here small means 10 or less retrievals.

```
*SAMPLE R2

 FIRST CITATION FOUND IN 18 SECS

 CIT NUM 00304555
 BIENER K
 <DIFFERENT PROBLEMS OF DRUGS IN TRADE SCHOOL AND HIGH
 SCHOOL STUDENTS (AUTHOR'S TRANSL)>
 PRAXIS VOL62  1612-5  26 DEC 73
 *DRUG ABUSE(OCCURRENCE)          *STUDENTS
  ADOLESCENCE                      AGE FACTORS
  CANNABIS                         COCAINE
  COMPARATIVE STUDY                ECONOMICS
  EXPLORATORY BEHAVIOR             FEMALE
  HUMAN                            LYSERGIC ACID DIETHYLAMI
  MALE                             MESCALINE
  SOCIOECONOMIC FACTORS            STATISTICS
  SWITZERLAND
*QUAL M13
 Q1=BLOOD
 Q2=CEREBR.FLUID
 Q3=CHEM.INDUCED
 Q4=CLASSIFICAT.
 Q5=COMPLICATIONS
 Q6=DIAGNOSIS
```

Q7=DRUG THERAPY
Q8=EDUCATION
Q9=ENZYMOLOGY
Q10=ETIOLOGY
Q11=FAMIL&GENET.
Q12=HISTORY
Q13=IMMUNOLOGY
Q14=INSTRUMENTATION
Q15=MANPOWER
Q16=METABOLISM
Q17=MORTALITY
Q18=NURSING
Q19=OCCURRENCE
Q20=PATHOLOGY
Q21=PHYSIOPATH.
Q22=PREV CONTRL
Q23=RADIOGRAPHY
Q24=RADIOTHRPY
Q25=REHABILITAT.
Q26=STANDARDS
Q27=SURGERY
Q28=THERAPY
Q29=URINE

The following three combine commands show how to build up a search statement which will find references if they are indexed under "drug abuse" linked to either of two qualifiers. Note that a match will not occur unless one of these qualifiers is actually specified for the main heading "drug abuse", and that terms may appear several times in one citation linked to different qualifiers.

*COMBINE M13 LINK Q19

R3= M13 LINK Q19
EXPECTED RETURN: SMALL

*COMBINE M13 LINK Q22

R4= M13 LINK Q22
EXPECTED RETURN: SMALL

*COMBINE R3 OR R4

R5= R3 OR R4
EXPECTED RETURN: SMALL

*SEARCH R5

APRIL AND MAY 1974 CITATIONS

FIRST CITATION FOUND IN 14 SECS

CIT NUM 00297727
LOWINGER P
HOW THE PEOPLE'S REPUBLIC OF CHINA SOLVED THE DRUG ABUSE PROBLEM.
AM J CHIN MED VOL1 275-82 JUL 73
*DRUG ADDICTION(PREV CONTRL) ATTITUDE TO HEALTH
CHINA DRUG ABUSE(PREV CONTRL)

DRUG ADDICTION(DRUG THERAPY)
DRUG AND NARCOTIC CONTRO
HISTORY OF MEDICINE, 19T
HISTORY OF MEDICINE, MED
HUMAN
OPIUM(HISTORY)
DRUG ADDICTION(REHABILITAT)
HISTORICAL ARTICLE
HISTORY OF MEDICINE, 20T
HONG KONG
MORALS

CIT NUM 00305850
BRIGGS AH
CAN WE PREVENT DRUG ABUSE IN INDUSTRY?
TEX MED VOL70 49-54 JAN 74
*DRUG ADDICTION(PREV CONTRL)
DRUG ABUSE(PREV CONTRL)
HUMAN
*INDUSTRIAL MEDICINE
HEALTH EDUCATION
UNITED STATES

CIT NUM 00287976
EINSTEIN S
DRUG ABUSE TRAINING AND EDUCATION: THE COMMUNITY ROLE.
AM J PUBLIC HEALTH VOL64 99-106 FEB 74
*DRUG ABUSE
ATTITUDE OF HEALTH PERSO
CURRICULUM
DRUG ABUSE(PREV CONTRL)
HUMAN
METHODS
SCIENCE
UNITED STATES
*HEALTH EDUCATION
CRIME
DECISION MAKING
DRUGS
JURISPRUDENCE
RELIGION
SOCIAL VALUES

CIT NUM 00234363
DISTASIO C NAWROT M
METHAQUALONE.
AM J NURS VOL73 1922-5 NOV 73
*DRUG ABUSE(PREV CONTRL)
ADULT
DRUG WITHDRAWAL SYMPTOMS (DRUG THE)
JAPAN
UNITED STATES
*METHAQUALONE
DRUG AND NARCOTIC CONTRO
HUMAN
PENTOBARBITAL(THERAP USE)

OCTOBER 1973 TO MARCH 1974 CITATIONS

CIT NUM 00244640
GREENE MH DUPONT RL RUBENSTEIN RM
AMPHETAMINES IN THE DISTRICT OF COLUMBIA. II. PATTERNS OF ABUSE IN AN ARRESTEE POPULATION.
ARCH GEN PSYCHIATRY VOL29 773-6 DEC 73
*AMPHETAMINE
*DRUG ABUSE(OCCURRENCE)
ADULT
DRUG ABUSE(PREV CONTRL)
FEMALE
HUMAN
METHADONE
*CRIMINAL PSYCHOLOGY
*SOCIAL CONTROL, FORMAL
DISTRICT OF COLUMBIA
DRUG ADDICTION(OCCURRENCE)
HEROIN ADDICTION (OCCURRENCE)
MALE
VIOLENCE

CIT NUM 00260832
REDFIELD JT
DRUGS IN THE WORKPLACE--SUBSTITUTING SENSE FOR SENSATIONALISM.
AM J PUBLIC HEALTH VOL63 1064-70 DEC 73

```
*DRUG ABUSE                          *INDUSTRIAL MEDICINE
 ADOLESCENCE                          ADULT !!

ATTENTION INTERRUPT
 CANNABIS                             DRUG ABUSE(DIAGNOSIS)
 DRUG ABUSE(OCCURRENCE)               DRUG ABUSE(PREV CONTRL)
 DRUG ABUSE(URINE)                    DRUG ADDICTION
 HALLUCINOGENS                        HEALTH EDUCATION
 HUMAN                                HYPNOTICS AND SEDATIVES
 LYSERGIC ACID DIETHYLAMI             OCCUPATIONAL HEALTH SERV
 OPIUM                                OREGON
 SMOKING(OCCURRENCE)                  STUDENTS

*SIGNOFF

END SEARCH NUMBER 1
ELAPSED TIME WAS  38 MINS  23 SECS
```

2.5 Feedback

The type of interaction supported by almost all operational information retrieval systems can be summarized in this way. When commanded to do so, the system will display references, sometimes with details such as index terms or abstract. Having seen this type of information, the user decides whether the computer's response is relevant, and whether a change in the search statement is necessary. He may construct a new request, perhaps a modification of a previous attempt, and command the machine to search again. It is the user's responsibility to arrive at a satisfactory profile. The system may help him to find suitable words and it may do useful clerical tasks for him (we have seen that Medusa, for instance, keeps a record of terms and intermediate search statements with short codes for easy reference). However, there is no way in which users can inform these systems of their success in displaying relevant references, and the systems have no way of making use of such inform-

ps, to monitor it for the benefit of system designers and evaluators). Whatever the difficulties of formulating a theory of relevance for information retrieval might be, there is an obvious subjective definition for it. A reference is relevant if, on the basis of available evidence, the searcher recognizes it as the sort of reference he was looking for. If we could use his relevance judgements to influence the search, we would depend even less upon his ability to express his interest.

There are, in fact, at least two large, computer aided services which have experimented with relevance feedback, neither of which provides the end-user with on-line access to the files. A study was conducted by UKCIS - the United Kingdom Chemical Information Service - (Barker et al,1972). In analyzing search failures resulting from profiles constructed by users, they found that a disturbing number (34% of precision failures and 46% of recall failures) were attributable to faulty original statements of interest. An iterative process involving a fixed file was used to develop a profile. The first search was done with a profile devised with the aid of UKCIS staff. Abstracts of documents retrieved were assessed by the user and terms occurring in them given weights accordingly. Terms with weights above a certain threshold were then used to retrieve more references which were sent to the user for assessment. The process continued until no new relevant documents were found, then the resulting list of terms with high weighting were considered by the user, who could make changes to his

profile accordingly. The new profile would then serve his regular current awareness needs. The difference in performance between original and new profiles was estimated: precision remained about the same, but recall increased by up to 30%.

The other large scale use of relevance feedback is reported by Vernimb & Steven(1973). ENDS (European Nuclear Documentation Service) maintains a very large data base; 15% of the references (i.e. 200,000) are available on-line to ENDS staff. A query is formulated interactively, using Boolean strategies, and a small sample of the documents retrieved are checked for relevance. All the terms assigned to the documents in the sample are given weights (which may be negative) reflecting their postings to relevant and non-relevant references. In the batch processed search of the whole file, document weights are calculated using the term weights so obtained and those with weights above a threshold value are retrieved and ranked.

The major experimental work on relevance feedback has been done on the SMART system; the techniques are described by Ide & Salton(1971), and evaluations and further details can be found in several chapters of Salton(1971). SMART is a system in which a number of processing options can be specified independently of each other; the number of combinations that can be tried is enormous. Generally speaking, documents entering the system are characterized by lists of weighted terms derived from the text. Queries are processed in a similar way and a correlation function is specified

which measures the "distance" between a query and a document, two documents, or two queries. Retrieval consists of ranking the collection of documents according to their correlation with the query, choosing some cutoff point, and selecting the documents above it.

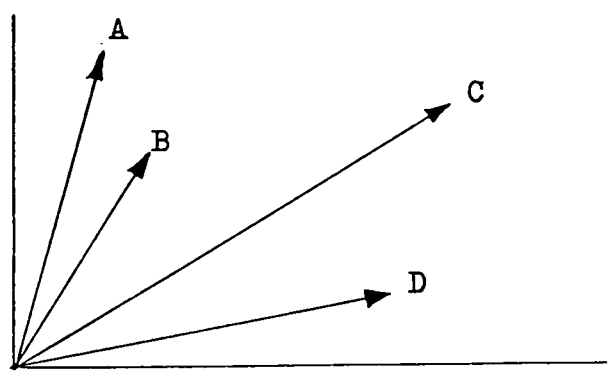
If the characteristics of the relevant documents retrieved are denoted by the vectors $\underline{r}_i$ and those of the non-relevant documents retrieved are $\underline{s}_i$, a query denoted by the vector $\underline{q}_0$ is updated by the equation:

$$\underline{q}_1 = \underline{q}_0 + \alpha \sum_{i=1}^{n_r} \underline{r}_i - \beta \sum_{i=1}^{n_s} \underline{s}_i$$

where n_r and n_s are respectively the numbers of relevant and non-relevant documents retrieved in response to query $\underline{q}_0$, and α and β are experimental parameters. In words, the terms occurring in relevant documents have their weights increased by α times the sum of their weights in those documents (new terms may be introduced). Terms occurring in non-relevant documents have their weights decreased similarly, but using β as the constant multiplier (terms whose weights become non-positive are deleted). The retrieval process is repeated with $\underline{q}_1$, and relevance decisions can, of course, be fed back as many times as the experimenter wishes. The constants α and β determine the extents of "positive" and "negative" feedback, respectively. It has been found that negative feedback is necessary for best performance. The majority of the improvement in performance comes with the first and second iteration, and can be substantial. A striking example of the effect of

feedback is given in Salton's second comparison of SMART and MEDLARS (Salton,1972).

A problem arises with the technique as a consequence of the method of retrieval (which is called "linear associative"). Whereas, with Boolean searching it is possible to construct a query that will retrieve any subset of a collection, so long as every item is indexed uniquely, it is not always possible to find a query that will bring any given subset to the head of the list with linear associative retrieval. We can illustrate this with a collection of four documents, labelled A, B, C, and D, and indexed by a vocabulary of 2 terms (weighted). The collection can be represented by a set of two--dimensional vectors:



If we define the distance function to be the angular separation of a pair of vectors, then no query can be found to retrieve just A and C, for instance, no matter how many iterations of query modification are executed. Any query that retrieves A and C must also retrieve B. The SMART system can, in principle, cope with this situation by clustering the document representatives and applying the query, and subsequent iterations, independently, to each cluster to which it is "close". The effect

of relevance feedback on the query will differ from one cluster to another, so that several different queries may be generated. To reach all the important documents, it may be necessary to consider many clusters.

3. Summary

We have covered a considerable amount of ground in this chapter, rather briefly of necessity. We started by trying to state the problem of reference retrieval, because the work reported in later chapters was motivated by the need to find practical solutions. If Kunz & Rittel (1972) are right when they say of this type of problem that "problem formulation is identical to problem solving", our attempt at stating the problem is bound to have failed. However, one needs a point of view from which one can define goals and refine them as sets of subgoals, and the nature of one's approach determines how far this refinement can be taken. The decomposition of a goal into subgoals can be regarded as an interpretation of the meaning of the goal. A goal has no practical meaning if it cannot be decomposed into achievable subgoals. If one can find no way of doing this, then one must either change one's point of view so as to avoid the problem, or accept some modification of the goal which can be achieved. If we do the latter, we are really changing the problem and must make sure that the new problem is a useful one to solve in the context of the old one.

We are continually coming up against such intractable goals: matching concepts, defining relevance, understanding natural language. Even the major goal - to build a

reference retrieval system - is ill-defined. The evaluation techniques used to assess the success of systems are indications of the goals that the designers are trying to achieve. The traditional measures of performance are recall, precision and fallout (the proportion of non-relevant material retrieved). They are calculated from the result of dividing a collection in two different ways, according to: (i) whether retrieved or not, (ii) whether "relevant" to the query, or not. The corresponding goal is to divide the collection, using the system, along the same line that human "relevance" judges would split it. That is, of course, a corruption of the goal which we tried to express in the first paragraph of this chapter. The objection to the modified goal is that there is no unique relevant subset of the collection. This is true even if the relevance judge is the man who needed the information. We have already remarked that the order in which he sees the references will affect the composition of the relevant subset. (In this connexion, there are some interesting discussions on evaluation in Cooper 1973, Vickery 1973, and Cleverdon 1974). The requirement to set up the very difficult goals is modified substantially in our favour when we can use a computer interactively. Decisions demanding intelligence are now achievable - the man makes them. As for the machine, we must find out how it can best select and present information, concerning which the man shall make decisions.

Chapter 3

INFORMATION HEURISTICS

There is, at present, no prospect of creating an automatic system capable of making an accurate record or representation of a researcher's information requirement, and delivering to him literature, of which it can be said, with confidence, that it will satisfy the need. We have seen that the fine discriminating powers of the searcher himself might be efficiently integrated with the crude powers of a machine in a well designed interactive system. When considering computer-aided literature searching, it is as well to keep in mind the fact that the final result of a search, from the user's point of view, is a set of documents judged by him to be relevant. Whatever device is used to inform him of references, and however much of the total search process is delegated (to machine or librarian), the end-user makes the final choice, rejecting the irrelevant. To return to figure 1 in Chapter 2 (repeated here as figure 4, for convenience), the set of

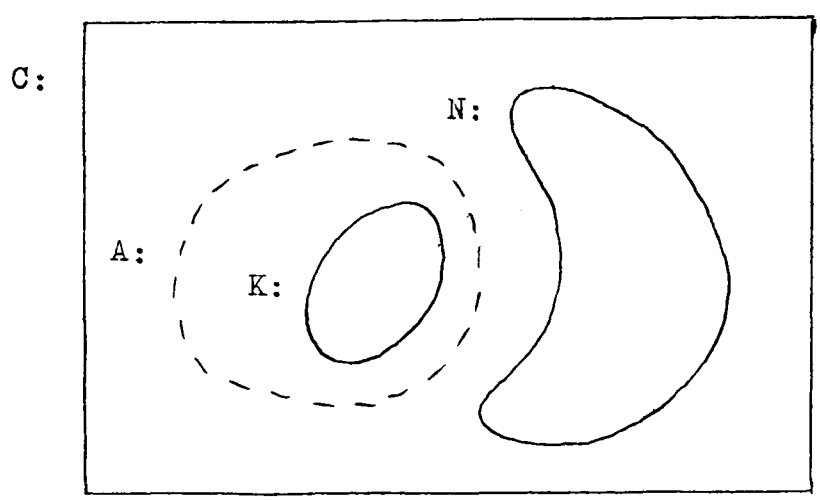


Figure 4.

references retrieved is a set $K \cup N$; the user will decide which belong to K (known to be relevant) and which to N (known to be not relevant). The success of the search depends upon the extent to which K meets the searcher's expectations (A). It is important to notice that as soon as the man has made a decision, the "uncertainty" that we introduced to discuss the problem of totally delegated searching disappears (i.e. $K \subseteq A$ and $N \cap A = \emptyset$).

Our aim, now, is to produce a mechanical aid to decision making, or problem solving, for the particular task of bibliographic searching. The decisions that the searcher wishes to make concern documents. The program developed here shows him references to documents, one at a time, and invites him to assess their relevance: in other words the sets K and N are specified by enumeration. The search should be efficient, in that the user should not have to consign many references to the set N; and at the same time it should promote awareness of what exists by permitting browsing among document surrogates (or ideally the documents themselves).

1. Dialogues for reference retrieval

Few scientists rely on a single source to provide all the information they need in their work. Among the most efficient and most frequently used sources is consultation with colleagues or subject specialists. In the course of such a conversation, some information will be exchanged - facts, opinions, and so on - and very often a few references to literature on the topic. Menzel has

written quite extensively on the role of personal communication in science (e.g. Menzel,1967). Regarded as a reference retrieval device, a subject expert is usually very precise. The process of retrieval by an expert has been studied by Olney(1962), who divides it into three steps:

> "1) interpreting the request as a statement of some kind of problem;
> 2) thinking up possible solutions to that problem;
> 3) selecting certain documents as relevant according to the contribution which information contained therein is likely to make to the more promising of these solutions." - p10.

Olney very optimistically thought that a large bibliographic retrieval system could be built on this basis. No mechanical subject expert has emerged so far. However, Olney's paper draws attention to an important aspect of dialogue which is generally missing from "man-machine dialogues", on the machine's side, at any rate. Each participant in a conversation tries to construct a model of the other's interest, in terms of his own view of the world. This is the basis of the first step, above, and it is the conceptual basis of the program, Thomas, described in this and the next chapter. Our machine's world-model is very simple, and cannot be used, by the machine, for problem solving. It is a set of document references embedded in a verbal context; the details will follow a little later.

The objectives of the program are to build a model, which must be defined in terms of its "knowledge", to show the man parts of it, and to use his reactions to bring the model into closer resemblance with the man's current interest. A set of functions with some suitable,

corresponding commands is the traditional approach to designing interactive computer software, but for this program it seems inappropriate. Firstly, although the knowledge structure that we have given our program is straightforward in principle, it is potentially very large. If its full richness is to be used effectively, we cannot put its manipulation under the direct control of the man; that would require huge feats of memory on his part. Secondly, we wish to build a program which is of use to the searcher who cannot specify what, precisely, he wants and would, thus find it difficult to issue commands. Actually, we have stated the view that, in general, it is not possible for a user to specify exactly, in advance, the attributes of relevant documents. For this reason, the user of program Thomas does not formulate a query.

The searcher starts by mentioning one or more points of interest - a subject, a document title, or an author. The program locates corresponding points in its "knowledge" of the literature and forms the initial model of the user's interest. The contents and various properties of the model are then used to determine a response to the user: the program usually tries to show him a reference together with some words and phrases descriptive of the document's subject matter. In his reaction, the searcher may assess the relevance of the document, indicate aspects of the description in which he is particularly interested, or not interested, and introduce new words. Thomas uses his assessment and any other inputs to update its model, and the cycle is repeated.

We shall go into the goals which the program tries to achieve, and its methods in a later section of this chapter.

From the user's viewpoint, a dialogue with Thomas is a browse through a collection of document surrogates, in which he may take whatever, and as much, initiative as he wishes. The subjects covered by the collection are seen by the searcher through their use in describing individual documents.

2. Modelling the user's interest

Our program creates and continuously adjusts a model of the area of interest of the enquirer. The model is constructed out of parts of the program's stored data about the literature. We do not consider in this thesis techniques for incorporating new information into the data base as a by-product of dialogues: this is not a learning program, though that might be a fruitful next step. A detailed description of the program is contained in Chapters 4 and 5. In this chapter, we shall try to give a view of the program from a higher level, so that reasons for the various design decisions can be seen. Let us start with the "data base", which is the program's knowledge of the literature.

2.1 The knowledge base

We are not attempting, here, to make any contribution to the art of indexing, or document description, so we adopt the familiar pattern of associating index terms and author's names with document references, and index terms

with each other. If we think of documents, subject descriptors and authors as being points in space, the association between two entities is a line joining the corresponding points, to signify that they are, in some sense, close. In principle, we need not restrict ourselves to associating document and author, document and subject, and subject and subject. We know from experience in conventional manual literature searching that document-document, author-author and author-subject associations are all useful, and would be valuable assets in the machine's data base. Neither should we necessarily restrict our consideration to documents, subjects and authors: corporate bodies and projects have their place in our knowledge of the literature of a field.

Associations between entities or ideas are themselves classifiable into various types, as are the associations between words or other symbols. Semanticists, philosophers, psychologists, computational linguists and librarians have all discussed the nature of the associations, but we shall side-step the issue, and say that two symbols are associated without trying to define the type of relationship that exists between them. This is acceptable in our retrieval methods, just as it was in L.B.Doyle's proposal, because "instead of depending on his [the user's] imagination to think up a search request, he is depending on his recognition of semantic relationships." (Doyle,1961,p577). Whitehall(1974) has described a successful manually maintained, growing thesaurus for an industrial research library. Associations between terms are plentiful, but it is left to the user to decide on the nature of the

relationships.

The program's "knowledge" is in the form of a network with labels on its nodes. Those that are associated are joined by lines. The important labels are references to documents, e.g.

> "Medullary carcinoma of the thyroid gland. A clinicopathologic study of 40 cases." Gordon et al, Cancer,31,pp915-24,Apr.73.

Other labels are names of authors, e.g.

> P.R.Gordon

and subject terms, e.g.

> thyroid neoplasms

We treat names just like subjects in manipulations of the network. The relationship of a name to a document may be that of "authorship" or "editorship", for example, and we may think of a subject being related to a document by "aboutness"; however, the program knows nothing of the types of these relationships.

In the discussions and desciptions which follow, it would be useful to have a small example network for illustrative purposes. Unfortunately,even for very small collections of references (20-30), the network is extremely difficult to draw: firstly, because if it is to be at all useful, it should be well connected, and one is then confronted with a figure resembling a seriously malformed spider's web; secondly, the labels on the nodes are long, and must be listed separately from the nodes themselves, thus making the associations difficult to appreciate at a glance. Nevertheless, we shall describe a small collection in sufficient detail for manipulation later.

(i) Example collection: "IR collection"

15 references from volume 16, 1973, of the Communications of the ACM. Indexing derived from that published with the papers.

Ref.1 "On Harrison's substring testing technique" A.Bookstein.
string, substring, hashing, information storage and retrieval

Ref.2 "Some approaches to best-match file searching" W.A.Burkhard, R.M.Keller.
matching, file organization, file searching, heuristics, best match

Ref.3 "On the problem of communicating complex information" D.Pager.
complex information, information, communication, mathematics, proof, language

Ref.4 "Hierarchical storage in information retrieval" J.Salasin.
information storage and retrieval, hierarchical storage

Ref.5 "Optimum data base reorganization points" B.Shneiderman.
data base, reorganization, files, information storage and retrieval

Ref.6 "A note on information organization and storage" J.C.Huang.
data base, data base management, information storage and retrieval, information structure, file organization, storage allocation, tree, graph

Ref.7 "A generalization of AVL trees" C.C.Foster.
AVL trees, balanced trees, information storage and retrieval

Ref.8 "Evaluation and selection of file organization - a model and system" A.F.Cardenas.
file organization performance, file organization model, secondary index organization, simulation, data base, access time, storage requirement, data base analysis, data management

Ref.9 "Design of tree structures for efficient querying" R.G.Casey.
tree, information storage and retrieval, clustering, searching, data structure, data management, query answering

Ref.10 "General performance analysis of key-to-address transformation methods using an abstract file concept" V.Y.Lum.

hashing, key-to-address transformation, random access, scatter storage, information storage and retrieval, hashing analysis

Ref.11 "Comment on Brent's scatter storage algorithm" J.A.Feldman, J.R.Low.
hashing, information storage and retrieval, scatter storage, searching, symbol table

Ref.12 "A data definition and mapping language" E.H.Sibley, R.W.Taylor.
data definition language, data structure, data base management, file translation

Ref.13 "The reallocation of hash-coded tables" C.Bays.
reallocation, dynamic storage, hashing, scatter storage

Ref.14 "A note on when to chain overflow items with a direct-access table" C.Bays.
hashing, open hashing, chaining, information storage and retrieval, collision

Ref.15 "Reducing the retrieval time of scatter storage techniques" R.P.Brent.
address calculation, content addressing, file searching, hashing, linear probing, linear quotient method, scatter storage, searching, symbol table

(ii) Index term list, with associations

There follows an alphabetical list of the terms used to index the IR collection. Most are linked to one or more of the above references, and to other index terms. The latter associations were made arbitrarily (but, it is hoped, sensibly) by the present author.

Term no.	Term	Assoc.refs	Assoc.terms
1	access time	8	21,29
2	address calculation	15	9,34
3	AVL trees	7	55
4	balanced trees	7	55
5	best match	2	38
6	chaining	14	8,17
7	clustering	9	22,25
8	collision	14	6,26,36
9	content addressing	15	2
10	communication	3	33,35
11	complex information	3	30
12	data base	5,6,8	13,14,24
13	data base analysis	8	12,51
14	data base management	6,12	12,15

Term no.	Term	Assoc.refs	Assoc.terms
15	data definition language	12	14,23
16	data management	8,9	45
17	data structure	9,12	6,19,32,52,55
18	dynamic storage	13	44,50
19	file organization	2,6	17,20,21,24,29, 43
20	file organization model	8	19,39,49
21	file organization performance	8	1,19,51
22	file searching	2,15	7,34,47
23	file translation	12	15
24	files	5	12,19,54
25	graph	6	7,55
26	hashing	1,10,11,13, 14,15	8,27,34,40
27	hashing analysis	10	26
28	heuristics	2	47
29	hierarchical storage	4	1,55
30	information	3	11,33
31	information storage and retrieval	1,4,5,6,7,9, 10,11,14	33,42
32	information structure	6	17
33	information system		10,30,31
34	key-to-address transformation	10	2,22,26,46
35	language	3	10
36	linear probing	15	8,38,40
37	linear quotient method	15	46
38	matching	2	5,36,47,52
39	mathematics	3	20,41
40	open hashing	14	26,36
41	proof	3	39
42	query answering	9	31
43	random access	10	19,46
44	reallocation	13	18,50
45	reorganization	5	16
46	scatter storage	10,11,13,15	34,37,43
47	searching	9,11,15	22,28,38
48	secondary index organization	8	
49	simulation	8	20
50	storage allocation	6	18,44,51
51	storage requirement	8	13,21,50
52	string	1	17,38,53
53	substring	1	52
54	symbol table	11,15	24
55	tree	6,9	3,4,17,25,29

It can be seen that there are many different types of association between terms, and no attempt has been made to

distinguish them.

(iii) "IR collection": reference table

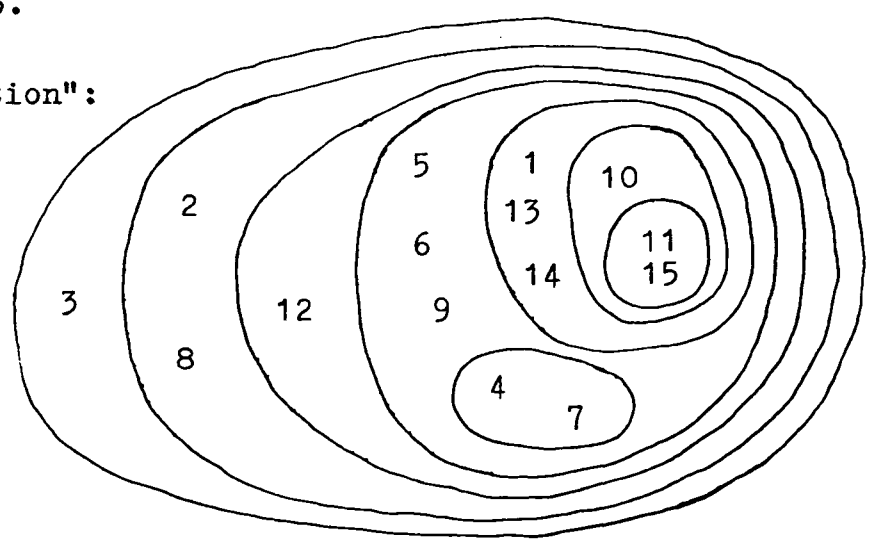
For convenience we list the documents with their descriptions, referring to the table of index terms, by number (the symbols appearing in this table are used in the later diagrammatic representations of graphs):

Doc.no.	Author(s)	Term nos
1	Bookstein	26,31,52,53
2	Burkhard, Keller	5,19,22,28,38
3	Pager	10,11,30,35,39,41
4	Salasin	29,31
5	Shneiderman	12,24,31,45
6	Huang	12,14,19,25,31,32,50,55
7	Foster	3,4,31
8	Cardenas	1,12,13,16,20,21,48,49,51
9	Casey	7,16,17,31,42,47,55
10	Lum	26,27,31,34,43,46
11	Feldman, Low	26,31,46,47,54
12	Sibley, Taylor	14,15,17,23
13	Bays	18,26,44,46
14	Bays	6,8,26,31,40
15	Brent	2,9,22,26,36,37,46,47,54

(iv) A collection-induced clustering of "IR collection"

We use the method given by Jardine & van Rijsbergen(1971) to generate a hierarchy of clusters of documents. The authors are regarded as index terms for this purpose. The principles of the method are explained in Chapter 2, section 2.3.

"IR collection":



Similarity between documents is strongest inside the inner-most ring, i.e. documents 11 and 15 are the "closest", and becomes weaker as we move outwards. The advantage of this type of clustering is that it forms the basis for an arrangement, in storage, of document references which can be used efficiently if one is content to limit the search to one, or a very small number (van Rijsbergen, 1974) of clusters. Van Rijsbergen claims that this is reasonable for collections in which the "Cluster Hypothesis" holds. Simply stated, this hypothesis is that documents which are relevant to the same query tend to have similar descriptions. This is a statistical phenomenon, which is more pronounced in some collections than in others (van Rijsbergen & Sparck Jones,1973). The design of the program Thomas makes use of such general properties; but it also takes account of the inevitable deviations, and we consider that this is an important feature of it. We shall discuss this last point again in the next section, and the example near the end of the chapter (section 4) illustrates it.

(v) "IR collection": part of the association graph

Figure 5 is part of the network in the neighbourhood of the seven documents, numbered 1,2,10,11,13,14 and 15. Documents 1,10,11,13,14 and 15 have been chosen because they are close to one another in the clustering given above; it can be seen from figure 5 that each of them is within a path length of 2 lines from all the others. Document number 2, on the other hand, is separated from the others in the hierarchical clustering. It's minimum "distances" from the other document nodes in the

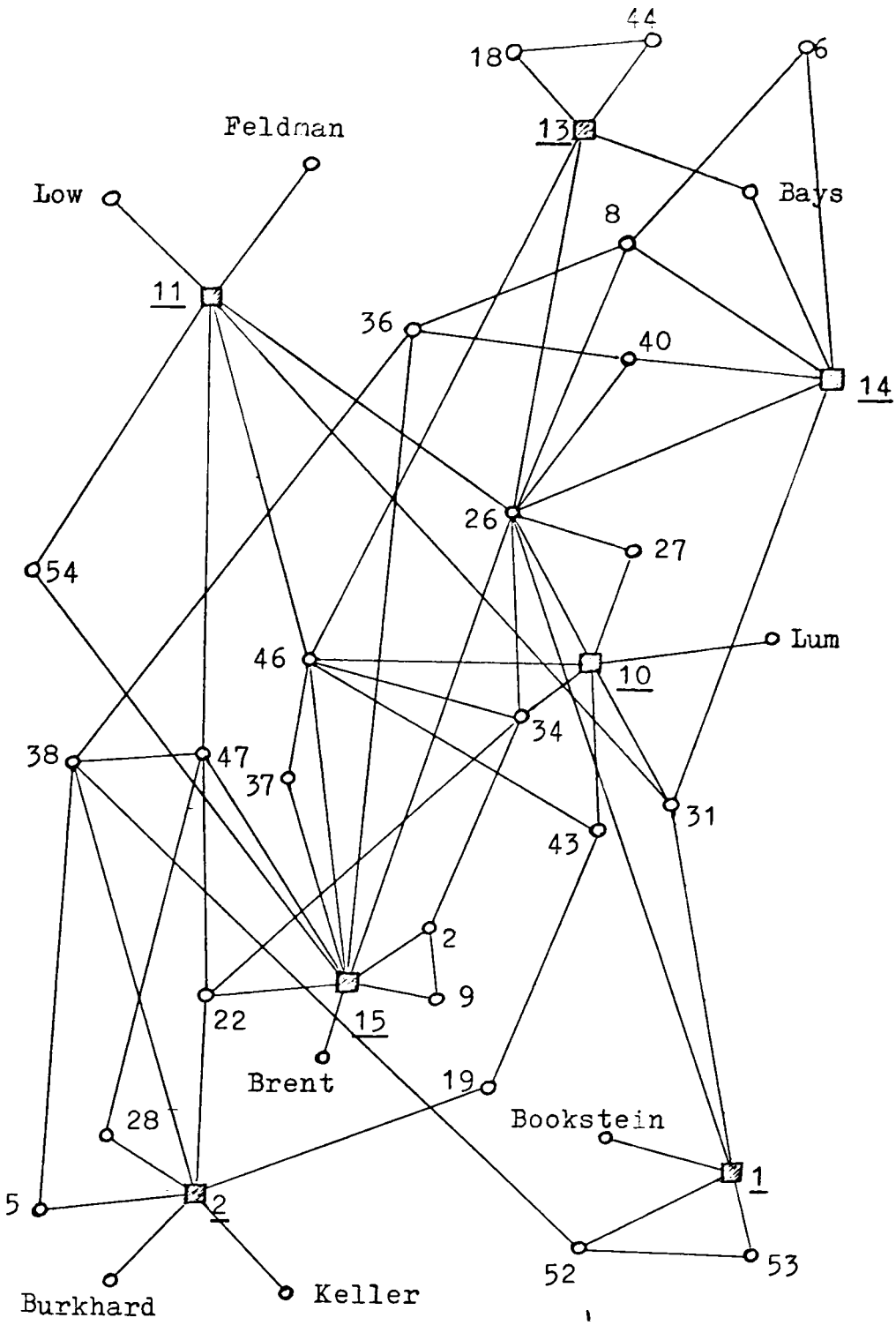


Figure 5. In this representation of part of the network, the dark squares stand for document nodes.

(sub)network shown are

(2,1) : 3 lines,
(2,10) : 3 lines,
(2,11) : 3 lines,
(2,13) : 4 lines,
(2,14) : 4 lines, and
(2,15) : 2 lines.

Document 2 is related to the others in a way that is not apparent in the description-induced clustering, but which can be found by a suitable search in the network. This is a very simple example of the sort of situation which Thomas can handle.

2.2 Retrieval by association

Retrieval by program Thomas is associative. The user indicates which labels interest him, and further labels (particularly references) are selected for his inspection from among those reachable by paths of association from the interesting ones. There are distinctions to be drawn between methods of the type proposed here, and the uses of association reviewed in Chapter 2, section 2.3. On the one hand, it has been suggested (Bush,1945) that associative links between items should be recorded in a machine to form "trails". As noted in Chapter 2, several computer-based systems have been inspired by the hypothetical "memex", as Bush called it. Treu(1970) describes such a system in detail: trails are given names for easy recall, and a recorded item may be placed in several trails. Retrieval is guided by the user, who tells the machine where to start and which trail to follow. He is shown item after item, can backtrack, and may select alternative trails. Similar facilities have been incorporated into a few text editors (Engelbart et al 1973,

van Dam & Rice 1970), so that the "on-line writer" may hop about his text, not constrained to think of it as sequential. The search is directed entirely by the user, and that is probably quite reasonable because, in these systems, he, or a close colleague, was the one to set up the trails. In a bibliographic network, the choice of trails available to him would be bewildering. What our program does is roughly equivalent to following many short trails in parallel, and, basing its decision on whatever hints the user has supplied, picking one of them to show him. The program has ways of blocking trails which the user does not like, and can retrieve material on many different trails.

Other uses of association in retrieval are based on statistical properties of the assignment of index terms to documents (Stevens et al,1965). A brief account of this area has been given in Chapter 2. Associations between items are calculated, using a statistic which measures their tendency to co-occur. Retrieval strategies which use links formed in this way generally use only strong associations: although the occasional weak link leads to important references, more often a great deal of irrelevant material would be retrieved. In the interactive search, the situation is different. A user can increase the importance of a tenuous association if he wishes. Statistically derivable associations of the types used by Jardine & van Rijsbergen(1971) to cluster documents, or by Sparck Jones(1971) to produce classes of keywords are obtainable from the network structure used by Thomas. We do not, however, work them out and record them explicitly

in the network: it is more useful to insert what we referred to as "semantic" links in section 2.3 of Chapter 2. The network used for trials of Thomas was obtained from a file of bibliographic records, which supplied document, author and subject nodes, and subject-document and author-document links. Subject-subject links were derived from a conventional thesaurus (ignoring the hierarchical direction of the links). Details can be found in section 2 of Chapter 7.

2.3 Model of context

The principle component of the program's model of the user's interest is called the "context graph". A simplified description of it would be that it contains nodes from the complete network (corresponding to items of various types - documents, authors, subjects) known to be of interest to the searcher, and a selection of nodes associated with those. Where two nodes in the context graph are joined by a line in the network, that line is inherited by the context graph. Nodes known not to be of interest are excluded. The second essential part of the model is thus a list of all the nodes which are known not to be of interest.

The goal of the program is to make the "context graph" a fruitful representation of the context of the user's enquiry. In other words, it should include the references which will satisfy him, and should have a structure which facilitates the selection of those references. The two components of the model that we have mentioned so far - the context graph and the set of

unwanted nodes - can be regarded, at any stage in the dialogue, as the current interpretation of the user's area of interest. Many of the program's heuristics require information about the history of the dialogue, and various sets of nodes and numerical values are considered to be parts of the model and maintained for this purpose. We shall introduce them as we need them in this chapter. Chapter 4, section 2 contains a more formal description of the model.

3. Creation and maintenance of the model

It would be as well to explain our use of the word "heuristic", in view of its common association with artificial intelligence studies and problem solving programs. We do not claim that Thomas solves problems or is in any way intelligent: it is the human user who must exercise his intelligence. Workers in machine intelligence describe a wide variety of programs as heuristic. Precise definitions of the term are hard to come by. Broadly speaking, it is applied to procedures which are based on the programmer's knowledge and common sense, but which are not guaranteed to complete, successfully, their assigned tasks (see, for example, Simon 1965, Minsky 1968, Science Research Council 1973). Often a program will contain more than one heuristic procedure for the same task - if the first fails, the next is tried, and so on. There are two main reasons for using heuristics: firstly, it may be that no deterministic algorithm is known for the required task; secondly, all known, complete solutions may be far too expensive.

The one solution to the reference retrieval problem which is sure to work is to present the whole collection to every enquirer, regardless of the query, and let him select the relevant documents. This is quite clearly a ridiculous approach, but we should remember that any more practical system, i.e. one which performs a preliminary selection or sorting, must employ heuristic procedures, because we do not know how a man makes relevance judgements. A feature of heuristic solutions is that it is usually not possible to characterize them as right or wrong; we can only make comparisons and state that one method performs better than another in certain respects. Some of the heuristics used by Thomas, to influence the state of its model and to respond to the searcher, have undergone several modifications and could, no doubt, be further improved. We beleive, however, that they perform sufficiently well to illustrate a viable approach to handling bibliographic data for information retrieval.

3.1 Using the model

Assuming that the program has formed a context graph like the one shown in figure 6, how should a reference be selected from it for consideration by the user? Document nodes (from "IR collection") are represented by black squares. The document nodes vary in their involvement in the context graph. Some, such as 6, 9, 14, are on the periphery: most of their neighbours in the complete network are not in the context graph. It seems sensible to use a measure of the involvement in choosing a reference. The other factor which we should

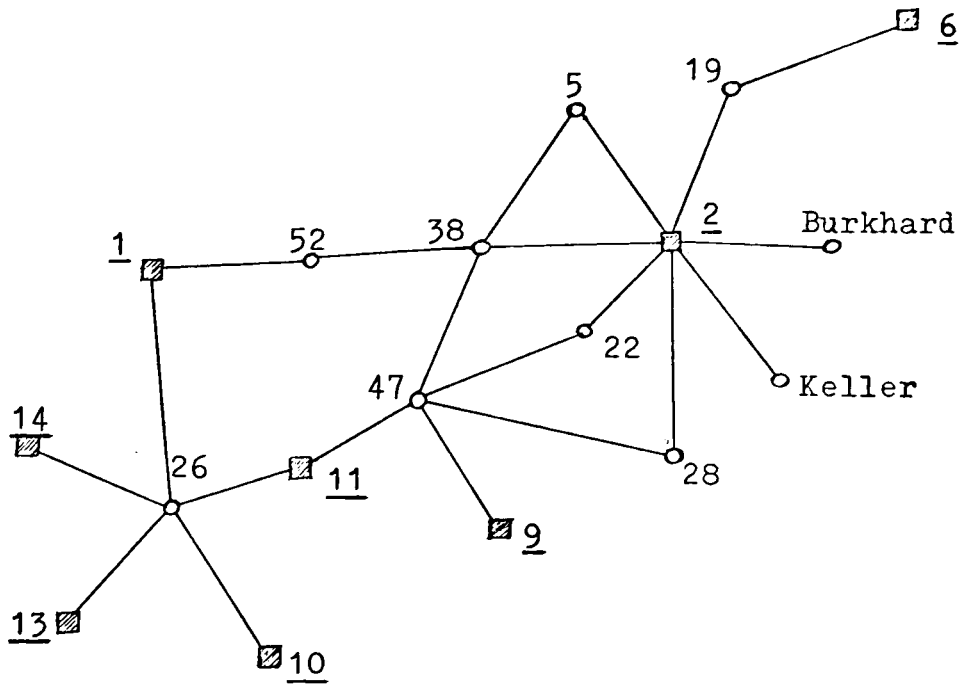


Figure 6. A context graph

consider is the degree of correspondence between the model and the user's interest. To gauge this, the program must observe the reactions of the user to what has already been shown to him. If the recent performance* of the program has been poor, according to the user, some special corrective action should be taken; but first we deal with the case where the program is performing reasonably well.

When we are prepared to accept that the context graph is a good representation of the field of enquiry, the program usually chooses the reference with the highest involvement in the model. Involvement of a node is measured by counting the number of nodes adjacent to it in the context graph, and dividing by the number of such nodes in the full network. Here are the values for each

---

* see Chapter 4, section 3.2.1

document node in the graph of figure 6:

Reference number	Involvement measure
2	1·0
1	·4
11	·286
13	·2
14	·167
10	·143
9	·125
6	·111

The user would be shown reference 2 - "Some approaches to best-match file searching" by Burkhard & Keller - unless he had already seen it earlier in the dialogue. To be precise, one should say that the program picks the most highly involved node which has not already been displayed, if such a node exists. This choice is intended to achieve the short-term goal of giving the searcher a relevant reference. If the reference selected proves to be of no interest, there are, at least, good prospects of being able to reduce the size of the context graph, because several nodes on display are in the context graph and may be eliminated as a result of the user's negative response.

A searcher who is collecting references during a dialogue with a computer is unlikely to want a large number of them. This provides us with a motivation for trying to keep the context graph small. We take what opportunity we can to delete nodes, and place limitations on the incorporation of new nodes. At various times, the context graph is inadequate as a source of relevant references, and more nodes must be added if the dialogue is to continue. The user may take the initiative by

spontaneously supplying a new subject term or author's name, for instance, but we should not rely on his ability to do that. The program will encourage growth of the context graph in the vicinity of nodes which are known to be of interest. References are not usually displayed more than once in a dialogue; but if the program's performance is unsatisfactory, or if the context graph contains no further document nodes, a reference in which the user has previously shown interest will be chosen and displayed again. In this case, the user is reminded that he has already seen the reference and is asked to reconsider it. We know that a searcher's criteria for judging the relevance of documents and the usefulness of subject terms are affected by the course of a search, so his response to the second occurrence of the reference may bring about a significant change in the context graph. If no reference is available for review, he will be shown a subject or name which he has entered or previously selected, together with all associated subjects or names. These actions on the part of the program seem to be the natural way to promote "course correction", and their effectiveness will now depend upon the use made of the man's responses to the displays. We discuss this in sections 3.2 and 3.3.

Returning to figure 6 for an example, let us suppose that the searcher has, at an earlier stage, approved reference 2, but that the dialogue is not proceeding so well now. The program displays reference 2 again.

Please reconsider this document:

Some approaches to best-match file searching.;
Burkhard et al, CACM 16, 1973.
1. W.A.Burkhard, 2. R.M.Keller, 3. best match, 4. file organization, 5. file searching, 6. heuristics, 7. matching

Whereas up to this point, the search may have been concentrated on the aspects represented by "best match", "heuristics" and "matching" (term nos 5, 28 and 38 in figure 6), the user may now consider it more profitable to look into "file organization" (term number 19). The effect of indicating this to the program is that several new subject nodes (such as "data structure", "file organization model", "files", "random access") and associated document nodes will be brought into the context graph, which will become much denser in the region of subject node 19.

We have mentioned two kinds of inadequacy in the model of the user's interest:

(i) The context graph contains no document nodes that the user has not seen,

(ii) There is an ill-defined lack of correspondence between the model and the query, as revealed by poor performance in the dialogue. The context graph contains too many nodes which are not of interest.

There is a third state of the model which we regard as unsatisfactory:

(iii) The context graph is not connected; that is, it contains pairs of points which are not reachable

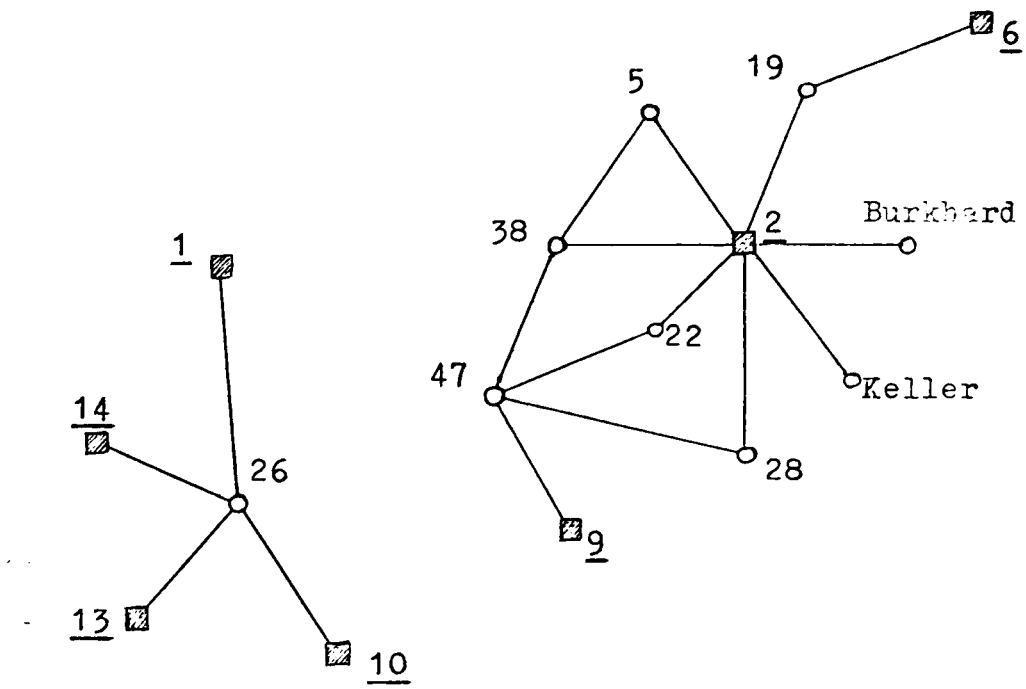
from each other by any path within the context graph.

It is assumed that the user is not attempting to conduct two or more totally unconnected searches at the same time. What is usually referred to as a "multi-aspect search" arises when the enquirer wishes to establish, or find, a link between ideas, or when he cannot express the concept that he has in mind in a single phrase, recognizable by the retrieval system. If our program can find a set of nodes in the network which form a bridge between otherwise unconnected parts of the context graph, these could lead to the retrieval of important references. We shall come back to this topic in section 3.3, and in Chapter 4, section 3.2.5, where a method for attempting to establish bridges is discussed. Here, we consider the selection of a document node for display, in the situation where the context graph is not connected.

A document node with a high involvement in the model would be "central" to just one of the connected components of the context graph, and we would expect it to be relevant to one aspect of the query, though not necessarily to be very useful in solving the searcher's underlying problem. At the other extreme, a document node having very low involvement is likely to be non-relevant. In Thomas, we opt, rather arbitrarily perhaps, for the reference with involvement closest to the average for all the unseen references in the context graph. The user may recognize a term which reduces, or even closes the gap between components.

As an example, consider again figure 6, and suppose

that subject node 52 ("string") and document node 11 have been rejected. The new context graph is shown in figure 7. In addition, we assume that references 1 and 2 have been displayed already. The two aspects of the model might be represented by the title of document 2 and index term 26;



component a component b

Figure 7.

namely, "Some approaches to best-match file searching" and "hashing". The references which are candidates for the next display, with their involvement measures, are:

Reference number	Involvement measure
13	·2
14	·167
10	·143
9	·125
6	·111
average	·149

The reference with involvement closest to the average is number 10, in component a. Two of its associated subject nodes (34 and 43) are also adjacent to subject nodes in component b. Term 34 ("key-to-address transformation") is linked to term 22 ("file searching"), and term 43 ("random access") is linked to term 19 ("file organization"). If neither of the terms 34 or 43 were acceptable to the user, these routes between the components would be blocked and attempts would be made to find another.

3.1.1 Document similarity

When a user judges a reference to be relevant, there are two obvious, sensible approaches to selecting the next reference. Firstly, we might say that the model is a good representation of the area of interest, and make another selection based on involvement, as described above. Secondly, we might assume that van Rijsbergen's Cluster Hypothesis holds, and find the document most similar, in terms of associated subject nodes, to the one just displayed, regardless of the context graph. The second method may select a reference which is not in the model. The program Thomas is capable of either procedure, or a mixture of the two.

The similarity measure finally used for the second method is the latest of a sequence of trial functions. It takes account of the searcher's expressed interests and, in a crude way, the usefulness (specificity) of index terms. The measure is given formally in Chapter 4, section 3.3.2. It is based on the extent to which documents share associated subject terms. Greater weight

is given to subjects which have been entered by the user, or selected by him from displays. Terms which he has rejected, but are nevertheless associated with the relevant document, are disregarded. Initially, no account was taken of the frequency of use of the terms in describing the collection (the term postings). A small number of very highly posted terms made nonsense of the similarity measure, however, and they are ignored by the final version of the similarity function unless explicitly mentioned by the user. The test collection for this project was derived from Medlars data, originally prepared at the USA National Library of Medicine. The indexers for that system consider a small number of common medical words, called check tags, for application to every document. A complete list of the check tags occurring in the test collection is given in Chapter 7, section 2. It includes, among others, HUMAN, MALE, FEMALE, CHILD and ANIMAL EXPERIMENTS.

Should we, then, use the normal procedure for picking a document node from the context graph, when the user has approved the last reference displayed; or should we use the similarity function? The main disadvantage of a similarity function of this type, for our purposes, is that it takes no account of the associations between terms. Information about the nature of associations is not recorded in the data base, but in the context of a particular search, a user may treat two terms as exact equivalents. The similarity measure ignores this possibility. On the other hand, similarity between documents will often be registered on the basis of terms to which the user is indifferent. If these terms are the only contributors to

the measure, we should not expect the "similar" document to be relevant, unless the value were particularly high. This suggests that we should only choose the document most similar to the one last displayed if the similarity measure exceeds a certain threshold. Otherwise, we pick a document node from the context graph. The formal definition of the procedure is given in section 3.3.2 of Chapter 4.

If the similarity threshold is zero, the similarity function will always be used after a judgement of "relevant" by the user; if it is (effectively) infinite, the similarity function will never be used. Experiments, like those described in Chapter 7, sections 4 and 5, in which the threshold was varied from one extreme to the other, indicate that the overall performance of the system varies only slightly with threshold value; neither extreme gave the best performance obtained. (The differences in performance are not statistically significant). To first try the similarity function, find that no document is similar enough to the one previously displayed, and then use the node involvement measure to choose a document, is rather an expensive procedure. We could remove the application of a similarity measure altogether without significantly degrading the retrieval effectiveness of the program.

3.2 Displays and messages

We have aimed for a very simple form of dialogue. The user's statements to the program are of one basic form, which is designed to be the vehicle for his response

to a displayed reference. In other circumstances, when no reference has been shown to him, a degenerate form of the statement is appropriate. The syntax of the user's statement has received little attention, and is very simple.

A message from the user is analyzed into three types of information, any or all of which may be absent. If he has been shown a reference, he may wish to say whether he is interested in it. He may also wish to single out certain aspects of the document description as being of particular interest, or definitely not of interest. Finally, he may have thought of a new term, author or title which may lead to further useful references. The display format of selected references is geared to these requirements. The label on the document node comes first, consisting of its title and information needed by the user to find the full document. This is followed by the labels of all the personal name and subject nodes associated with the document. They are numbered in the display so that the user may easily refer to them. Let us illustrate this with an example from "IR collection". Reference 1 would be displayed like this:

> On Harrison's substring testing technique.; Bookstein, CACM, 16, 1973.
> 1. A.Bookstein, 2. hashing, 3. information storage and retrieval, 4. string, 5. substring

Some responses that the user may make to this are as follows:

	Comments
(i) Yes	He is interested; we shall assume that all the numbered items are of interest.
(ii) Yes, 2	He is interested in the reference, and particularly in "hashing". We make no assumptions about the other numbered items.
(iii) Yes, Harrison	He is interested in the document, and presumably all the numbered items; and a new name is introduced, suggested to him by the title.
(iv) No	The reference is not relevant; none of the numbered items are of interest.
(v) No, 4	He is not interested in the reference, but "string" looks promising; the other items are assumed to be of no interest.
(vi) Yes, 1, not 2	The reference is relevant; particularly interested in author Bookstein; "hashing" is of no interest.
(vii) 4,5	He is making no comment about the reference, but is interested in "string" and "substring".
(viii) 'string matching', 'patterns'	He makes no comment about the reference or the numbered items, but introduces two new

terms.

(ix) [null message] The user makes no comment; he is indecisive and presumably wishes to see what will come up next.

The program's interpretation of the user's message is given precisely in Chapter 4, section 3.1. It should be pointed out here that assumptions in the "comments" column concerning numbered items which the user does not mention are simplifications, and are in fact modified by information which he has given earlier in the dialogue.

If, instead of displaying a reference, the program displays a group of subject terms, the responses "yes" and "no" are inappropriate, but otherwise the same statement form can be used. There are occasions when nothing has been displayed: (i) at the beginning of the dialogue, (ii) when all heuristics for selecting nodes for display have failed, and the program is forced to ask the user to take the initiative. The user must then supply one or more new names or terms; relevance judgements have no meaning.

A point to notice about the displays is that nodes which have previously been rejected by the user are not barred from appearing among the numbered items. The reason for this is our uncertainty of the status of these nodes. A user may say that the term "hashing" represents an aspect of a document which does not interest him. Nevertheless, there may be, in the collection, a useful document which touches upon the topic of hashing, incidentally so far as this user is concerned. Just as we cannot be certain that if a document is indexed by a

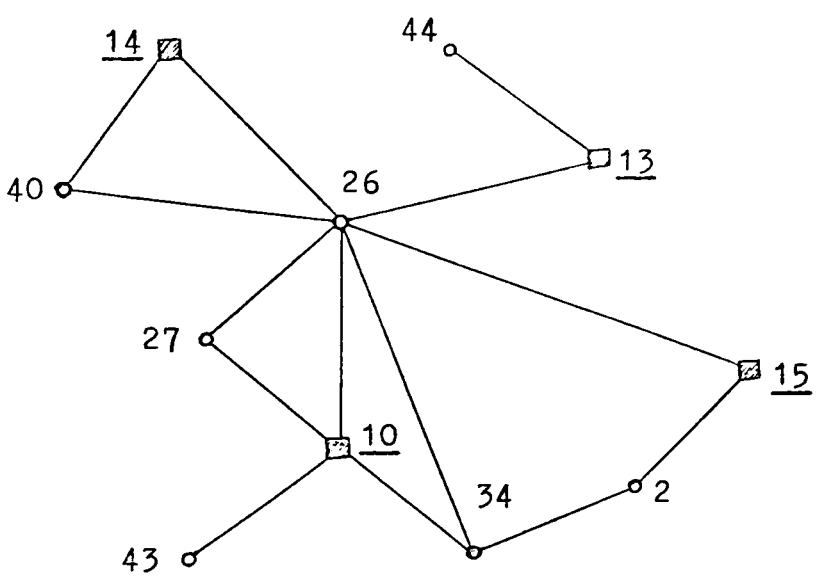
particular term then it is relevant, so, equally, we cannot be certain that the presence of any particular term implies that a document is not relevant. In fact, it is not at all unusual for a searcher to discover that a term, hitherto dismissed, is a useful hook for fishing out relevant references.

3.3 Modifying the model

Throughout this discussion, it should be remembered that the "deductions" that we can make from the user's messages are never very strong. We must be prepared for the user to change his mind. If growth of the context graph is inhibited in some region, it should not be too difficult to break through if the user appears to contradict his earlier statements. The assumptions made about the user's interests, as given above, are used to compile three sets of nodes (in the main network). Firstly, there may be explicit, textual requests in his statement, and the nodes with corresponding labels are found (details of this process are given in Chapter 5, sections 1.1 and 1.2). The second set contains all the nodes, represented in the last display, in which he is assumed to be interested, and the third set all those nodes in which, it is presumed, he is not interested. Not every item in the previous display is necessarily contained in one of these sets: there may be some, concerning which no assumption should be made. The sample responses listed in the section above illustrate cases of this type. We shall refer to the sets as "requested", "selected" and "rejected" nodes respectively.

The rejected nodes are used to determine where the context graph shall be "pruned", and where future growth shall be inhibited. Even nodes previously requested or selected may be removed from the context graph: the user must state, explicitly, that he is no longer interested in them. The removal of a node from the context graph brings about the removal of all lines incident with it. Thus, the context graph may become unconnected. Figure 8 is a particular context graph derived from the "IR collection". The document node most "involved" is number 10, so it is displayed with <u>all</u> its neighbours in the complete network:

> General performance analysis of key-to-address transformation methods using an abstract file concept.; Lum, CACM, 16, 1973.
> 1. V.Y.Lum, 2. hashing, 3. hashing analysis, 4. information storage and retrieval, 5. key-to-address transformation, 6. random access, 7. scatter storage



<u>Figure</u> 8.

There follows a table of correspondences between display identification numbers and subject node numbers:

display no.	subject node no.
2	26
3	27
4	31
5	34
6	43
7	46

It can be seen that some of the items in the display (1, 4 and 7) are not in the context graph. Let us suppose that, in response to the display, the user types:

not 2,4

It is assumed that he is not interested in subject nodes 26 and 31; no assumption can be made about the document node (10) or any other associated node. Subject node 26 is removed from the context graph, which becomes unconnected (figure 9). The other rejected subject node (31) is not in the context graph, but the program will remember that it has been rejected and will not allow it to join the context graph at any later stage, unless the user subsequently requests or selects it, i.e. changes his mind.

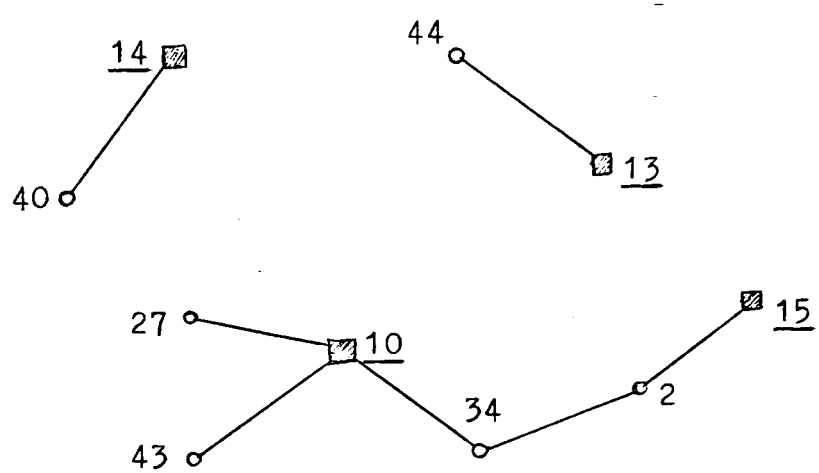


Figure 9.

nodes selected from the display by the user will be added to the context graph, if they are not already contained in it. If their use had previously been inhibited, it would no longer be so. As with nodes corresponding to the user's textual requests, selected nodes are given special status for use in future manipulations of the context graph, choices of nodes for display, and interpretations of the user's responses. In addition to the actual nodes selected, which are subject and author nodes, the program incorporates in the context graph any document node which is associated with a selected node and which is not already in the model. Now, this rule needs qualification. It was found that the "check tags" (see section 3.1.1) once more caused trouble. If a check tag is selected, and all its associated document nodes brought into the context graph, the model becomes very large and much of its bulk is irrelevant. Many documents are linked to several check tags, and could have a high involvement measure within the context graph purely on the basis of the check tags. The program, therefore, only incorporates in the context graph document nodes associated with selected nodes which are not check tags. We should make it clear that check tag nodes may occur in the context graph and be taken into consideration when calculating the involvement of document nodes. They are not, however, used to bring new documents into the model.

The action taken with requested nodes is similar. The nodes themselves are included in the context graph, and so are all non-inhibited nodes, whatever their type, which are associated with those among them that are not

check tags. If, for example, the request is for "searching", the nodes incorporated in the context graph (from "IR collection") would be document nodes 9, 11 and 15, and subject nodes "file searching"(22), "heuristics" (28), "matching"(38) and "searching"(47). A result of the method of handling requested check tags is that if the user's initial request is for one of these very highly posted terms, then the program responds in very much the same way as a man would - it will not attempt to refer to the literature until a more specific request has been made. Suppose, for instance, that "information storage and retrieval" is a check tag: it is the most highly posted term in "IR collection". If the user types just that term, the context graph created is simply the single node:

Since there are no document nodes to choose from, Thomas will try to stimulate the user to give more topics of interest, with the display:

> Consider these subjects:
>
> 1. information storage and retrieval, 2. information system, 3. query answering

When the searcher's statement has been interpreted and used to influence the model, the context graph is checked for connectedness. We have already argued the

case for trying to maintain a connected context graph (section 3.1). Before selecting a reference for display, the program attempts to join up the connected components of the context graph, if there are more than one of them, by incorporating new nodes from the network data base. The method used by Thomas is described in section 3.2.5 of Chapter 4. Before any attempt is made to find paths between components (an expensive process), the context graph is examined with the object of discarding very small components of no particular interest. These are defined to be components of less than three nodes, none of which have been requested or selected by the user. Such components are usually separated from the main body of the context graph when rejected nodes are deleted. In figure 9, two small components have been formed in just this way. If subject 44 ("reallocation") has been requested or selected by the user, but 40 ("open hashing") has not, the context graph would be reduced to that shown in figure 10. Reference number 10 has been displayed, and subject nodes 26 and 31 are inhibited from joining the context graph.

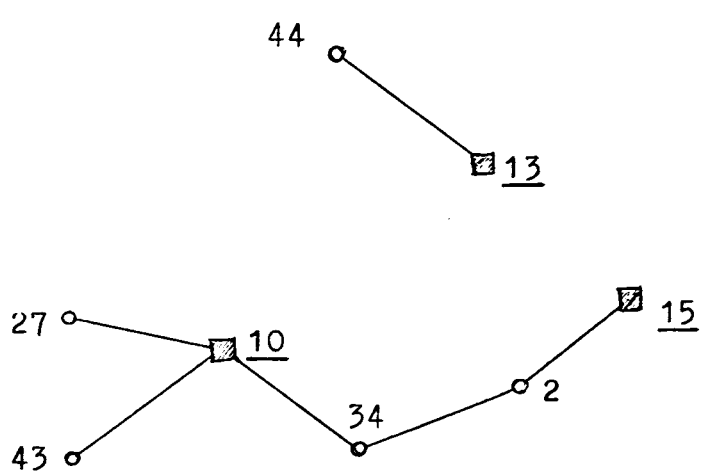


Figure 10.

The next step is to find a short path between the two components. The path length is restricted to two lines in our program; firstly to limit the amount of computation needed, and secondly to ensure a high likelihood of creating a useful bridge. The procedure employed starts by finding, for each component, the set of nodes adjacent to the non-check tags in the component, excluding inhibited nodes, and nodes already in the component. For the two components in figure 10, the sets are:

{Bays, 18, 46, 50}

and {Lum, Brent, 9, 19, 22, 31, 36, 37, 46, 47, 54} .

The numbers are all subject node numbers. These sets are intersected, and an element chosen from the meet, giving preference to document nodes. This element forms the bridge between the two components: it is associated with at least one node in each. Note that when there are more than two components to join together, it is not necessary to find a bridge between each pair. In our example the bridge must be subject node 46. This is added to the context graph in the same way as selected nodes are, i.e. accompanied by associated document nodes:

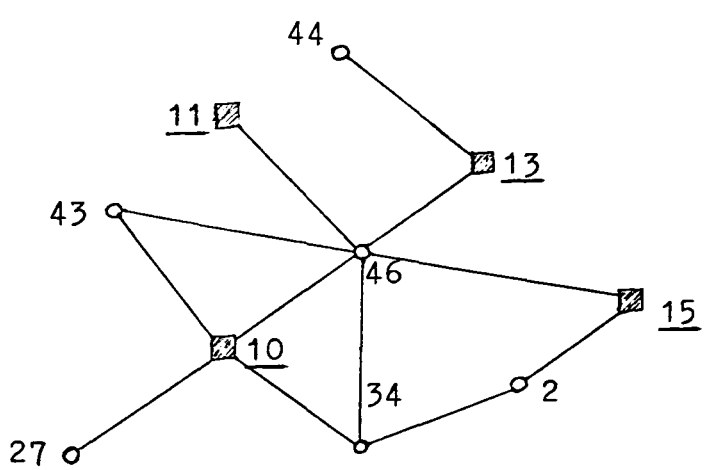


Figure 11.

The document node most involved in the context graph in figure 11 (after 10, which has already been displayed) is no. 13, so the next display will be:

> The reallocation of hash-coded tables.; Bays, CACM, 16, 1973.
> 1. C.Bays, 2. dynamic storage, 3. hashing, 4. reallocation, 5. scatter storage

4. A search (example)

We conclude this account with an example, using "IR collection". Any search in such a small collection is bound to appear artificial, or contrived. On the other hand, one can follow the processes easily. We add a little more information to the specification of the data base: the two most frequently posted terms ("hashing" and "information storage and retrieval") are designated check tags.

The search is for documents which may have a bearing on techniques for inexact matching of data. Let us say that the user will judge documents 1 and 2 to be relevant. A glance at the collection-induced clustering given in section 2.1 will show that these two documents are quite widely separated, and that a retrieval technique based on that clustering would not be satisfactory for this search. In fact, the two document descriptions have no subject terms or authors in common. The search which follows lacks realism largely because so many terms are associated with only one document.

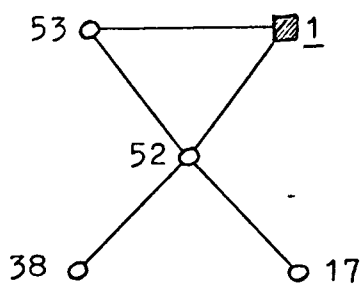
User: 'inexact string matching'

Thomas: Do you mean string?

The program has not been able to find a better phrase, and the user accepts 'string' as an aspect of his enquiry.

User: Yes

Thomas: Initial context graph is the star surrounding subject node 52 ('string'). Note that the aspects "inexact" and "matching" are not yet known by the program as being of interest.

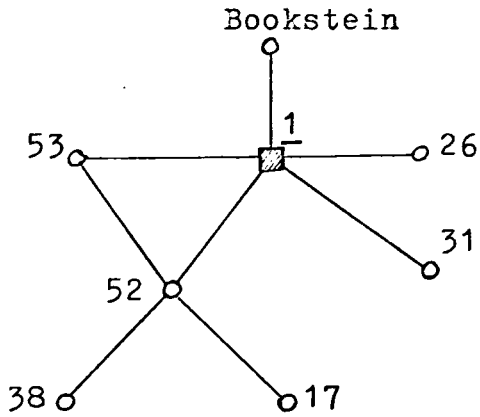


The most highly "involved" document node is chosen for display. (It is luck that in this case it is a relevant one).

On Harrison's substring testing technique.;
Bookstein, CACM, 16, 1973.
1. A.Bookstein, 2. hashing, 3. information storage and retrieval, 4. string, 5. substring

User: Yes

Thomas: All numbered items in the display are added to the context graph. No new document nodes are added, because subjects 26 and 31 (2 and 3 in the display) are check tags, and the other items are only linked to document 1.

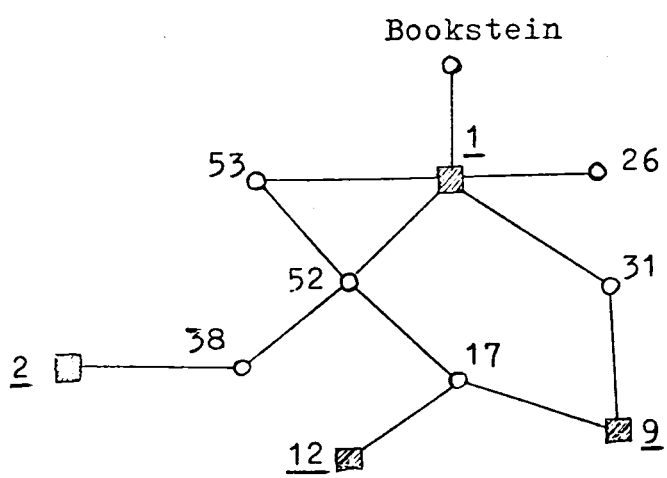


The program looks for a document (in the whole collection) similar to no.1 - and fails.
There is no other document in the context graph, so the program suggests subjects adjacent to 52 ('string'):

Consider these subjects: 1. string, 2. data structure, 3. matching, 4. substring

User: 2,3,4 He is interested in all of them.

Thomas: Subject nodes 17, 38 and 53 (2, 3, 4 in the display) are added to the context graph with their associated document nodes:



Involvement measures of undisplayed document nodes are:

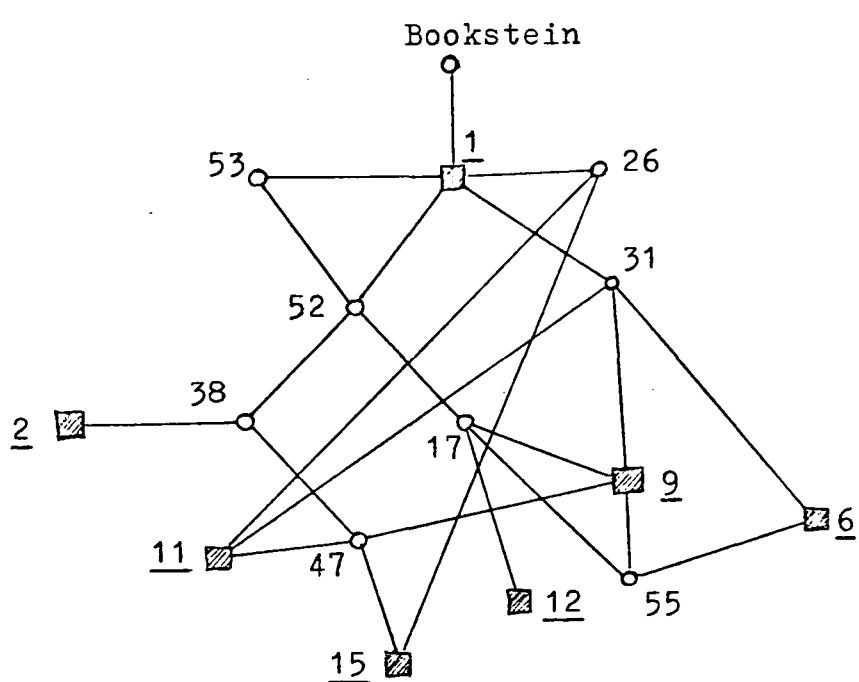
doc.9	·25
doc.12	·167
doc.2	·143

Document 9 represents the data structure aspect of the topic.

> Design of tree structures for efficient querying.; Casey, CACM, 16, 1973.
> 1. R.G.Casey, 2. clustering, 3. data management, 4. data structure, 5. information storage and retrieval, 6. query answering, 7. searching, 8. tree

User: 7,8 He is non-committal about the reference; 'searching' and 'tree' could lead to references.

Thomas: Subject nodes 47 and 55 (7 and 8 in the display) are added to the context graph with associated document nodes:



Involvement measures of undisplayed document nodes are:

doc.11	·429
doc.6	·222
doc.15	·2
doc.12	·167
doc.2	·143

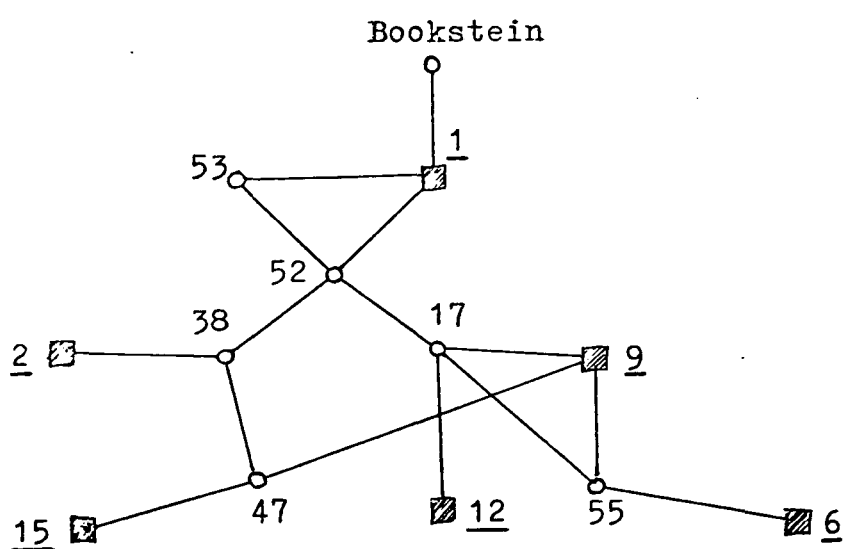
Now we follow up the hashing aspect of document no. 1, with document no. 11:

> Comment on Brent's scatter storage algorithm.; Feldman, et al, CACM, 16, 1973.
> 1. J.A.Feldman, 2. J.R.Low, 3. hashing, 4. information storage and retrieval, 5. scatter storage, 6. searching, 7. symbol table

(Note that, although 'hashing' and 'information storage and retrieval' have contributed to the selection of this reference, as check tags they played no part in bringing it into the model)

User: No

Thomas: Document node 11 and subject nodes 26 and 31 are removed from the context graph. Use of the following nodes will be inhibited: doc.11, names Feldman and Low, subjects 26,31,46,54.



Involvement measures:

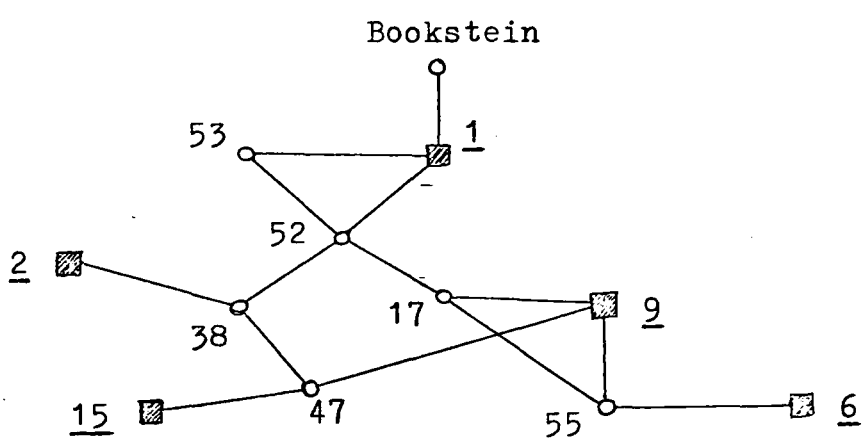
doc.12	·167
doc.2	·143
doc.6	·111
doc.15	·1

Note how the hashing aspect has subsided. We return to data structuring with document no. 12:

> A data definition and mapping language.; Sibley, et al, CACM, 16, 1973.
> 1. E.H.Sibley, 2. R.W.Taylor, 3. data base management, 4. data definition language, 5. data structure, 6. file translation

User: No — The only topic of interest is 'data structure', and he has already indicated that.

Thomas: Document node 12 is removed from the context graph. Use of the following nodes will be inhibited: doc.12, names Sibley and Taylor, subjects 14,15,23.



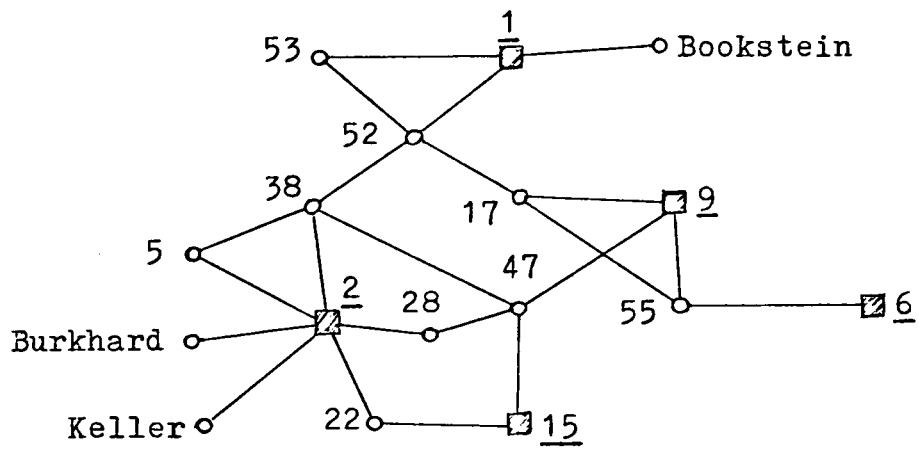
Involvement measures:

doc.2	·143
doc.6	·111
doc.15	·1

Some approaches to best-match file searching.;
Burkhard, et al, CACM, 16, 1973.
1. W.A.Burkhard, 2. R.M.Keller, 3. best match,
4. file organization, 5. file searching,
6. heuristics, 7. matching

User: Yes, not 4 This is the second relevant document.

Thomas: All of the items in the display, except that numbered 4 (subject node 19), are added to the context graph, with associated document nodes. Subject node 19 is inhibited from further use.



We shall leave the dialogue at this point, and give a summary of the state of the model by listing the subjects in the context graph:

53	substring
52	string
17	data structure
55	tree
5	best match
38	matching
47	searching
28	heuristics
22	file searching

and the subjects whose use is inhibited:

14	data base management
15	data definition language
19	file organization

23 file translation
26 hashing
31 information storage and retrieval
46 scatter storage
54 symbol table

5. Summary

We have given, in this chapter, a description of a program, called Thomas, with which a man can conduct a dialogue, serving to assemble a set of references relevant to his problem in hand. The philosophy behind the design of the program has been discussed: the concepts of (i) a dynamic model of the user's interest, (ii) browsing among document surrogates rather than through an indexing language thesaurus, and (iii) thereby doing away with coherent query formulation. The program represents another approach to the integration of man and machine in one system.

In the next chapter, the rather informal description given above is complemented by a more precise definition of the important functions of the program.

Chapter 4

FUNCTIONAL DESCRIPTION OF THOMAS

In this chapter, we give a detailed description of the reference retrieval program, without giving much attention to techniques or considerations of implementation. The program has undergone one major upheaval and several minor ones to reach its present state, but very little will be said about its history. Similarly there are many ways in which one could tinker with the program, none of which will be discussed here.

Broadly speaking, there are three components to the system: (i) the "data base", or bibliographic file, which is its stored knowledge of the literature, and is, for the present experiment, static; (ii) the model of the searcher's interest, which exists only for the duration of a search and develops as the dialogue progresses; (iii) the program, which uses the data base and the searcher's input to create and maintain the model, and uses that to select helpful references.

1. The "data base"

The bibliographic data which the program handles should be regarded as being attached to the nodes of an undirected graph. Let us call this the supergraph, because we shall frequently want to talk about parts of it (subgraphs and subsets of its nodes); it is a labelled graph.

Formally, the supergraph, S, is a triple (N,L,A), in which:

$N = \{n_1, n_2, \ \ldots \ n_p\}$, a set of p points,

$L = \{l_1, l_2, \ldots l_p\}$, a set of p labels, one for each point in N (i.e. there is a function, f, mapping N onto L; $f: N \rightarrow L$),

$A = \{\{n,m\} : n,m \in N$ and $\{n,m\}$ is prescribed and $n \neq m\}$, i.e. a set of unordered pairs of distinct points in N (not necessarily all such pairs) - the lines of the graph.

We shall be particularly interested in the sets, S_i $(1 \leqslant i \leqslant p)$, of points adjacent to each point, n_i, in N:

$$S_i = \{m : \{n_i, m\} \in A\}, \quad 1 \leqslant i \leqslant p.$$

1.1 Labels

The labels, l_1, l_2, etc., are bibliographic. Some stand for documents, and contain the type of information which usually occurs in a citation, some consist of the names of authors, and others stand for subjects or topics. In the data base under consideration all labels are derived from the bibliographic description of a collection of documents in the field of medicine and the indexing vocabulary associated with that (Medical Subject Headings from MEDLARS and synonyms from the Medusa system).

A label is structured data, or, in traditional terminology, a record. There are three types of label, distinguished by a type indication; they are as follows:

Type 1 (author label): contains a name (usually a surname) and initials.

Type 2 (document label): contains the title (a phrase), a reference to the document's location in, e.g., a journal (a character

string), and the "citation number" of the record in the MEDLARS file from which the label was derived (an integer).

Type 3 (subject label): contains a term or phrase.

With the exception of the citation number, all the components of the various labels are character strings of arbitrary length.

As bibliographic records go, our "labels" are exceedingly simple. Library cataloguing methods typically distinguish 50 "fields", from which an individual record may have a selection of some 20. The supreme example of complexity in record design in this area is surely the MARC (Machine Readable Cataloguing) record developed by the Library of Congress and the British National Bibliography (Gorman & Linford,1971). But that record structure was intended for an indefinitely large number of applications, and the label we are discussing is not. There are no more types of label nor subdivisions of data within labels than are required by the program.

Some examples of labels:

(i) author labels:

(name:"Hewetson", initials:"JF"),

(name:"Schulte-Holthausen", initials:"H").

(ii) document labels:

(title:"Distinct projections to the red nucleus from the dentate and interposed nuclei in the monkey",
reference:"Flumerfelt et al,Brain Res,50,408-14, 28 Feb 73",
citation number:144189),

(title:"<Systemic venous insufficiency. A new and rare syndrome>",

reference:"Groen et al,Phlebologie,25,399-406, Oct-Dec 72",

citation number:143603).

The angle-brackets in the second example indicate that the title is a translation from a language other than English.

(iii) subject labels:

"hemagglutination inhibition tests",

"rabbits",

"brain injuries, acute".

The way in which the collection of labels present in the experimental supergraph were chosen and obtained is described in section 2 of Chapter 7.

The mapping $f:N \rightarrow L$ mentioned above can be regarded as the "accessing function". The points n_i are "addresses" which the function f uses to access the labels l_i.

1.2 Lines in the supergraph

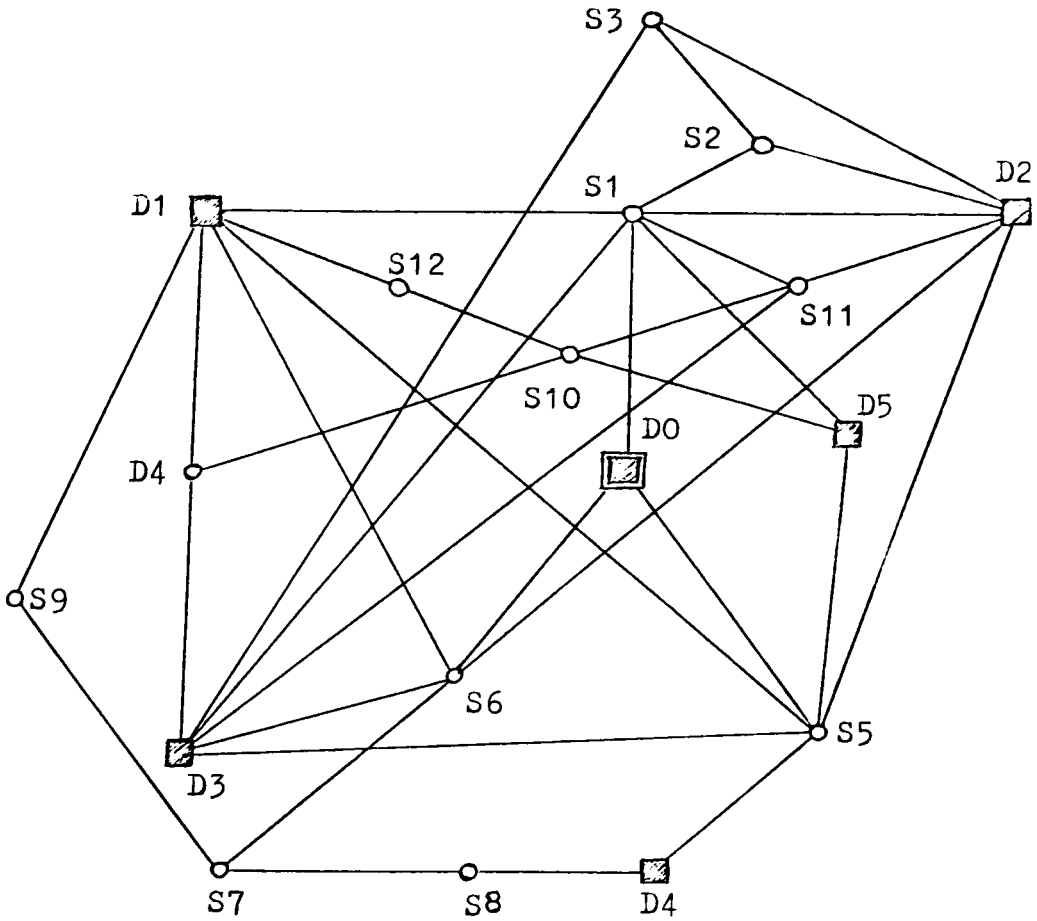
As the definition of A, above, implies, any distinct pair of points may be associated. There is no reference to the label set, L, and it should be noted, in particular that there is no restriction on the combinations of types of points that are linked (the type of a point is the type of the label attached to it). In the experiment, certain combinations happen to be absent, e.g. author-subject, but this should be regarded as a quirk in the data conveniently available for constructing the supergraph.

Unlike a point, a line has no label attached to it. Figure 12 is a pictorial image of some of the neighbourhood of a document node in the supergraph used for the experiments. Points and lines have their obvious representations. It should be remarked that a relatively sparse part of the supergraph was chosen for this figure, and even then ruthless pruning was necessary to produce a readily assimilable figure. 72 distinct points adjacent to those in the figure have been omitted (including all with author labels).

2. The model

The supergraph described above is the program's entire "knowledge" of the literature which a user may peruse. A model of an enquirer's interest developed by the program must be in terms of that "knowledge": something which is derivable from it, and which can be used to determine what, in the data base, should be shown to the user. In addition, it must be such as can be modified to reflect information gained from the user's responses. We shall now list the components of the model; further details on how they are maintained will be given later when the program's operation is described. The definitions which follow are in terms of the supergraph $\mathbf{S} = (\mathbf{N},\mathbf{L},\mathbf{A})$ - see section 1, above.

(i) context graph. This is an unlabelled subgraph of S. It is the maximal subgraph induced by a subset of the points in S. Formally, the context graph is a pair of sets $G_C = (N_C, A_C)$, where $N_C \subset N$ and $A_C = \{\{n,m\}: \{n,m\} \in A \text{ and } n,m \in N_C\}$. In other words,



Key (i) document labels (title parts only):

D0: "<Design of an evaluation questionnaire for pediatric nursing students>"
D1: "Toward defining the end product of medical education"
D2: "Reliability and validity of subjective evaluation of baccalaureate program nursing students"
D3: "Introduction of concepts of measurement and statistics to sophomore nursing students"
D4: "Quality-of-care assessment: choosing a method for peer review"
D5: "Evaluation of the American board of pediatrics oral examination by candidates after completing it"

(ii) subject labels:

S1:"Education measurement"
S2:"Faculty, nursing"
S3:"Students, nursing"
S4:"Curriculum"
S5:"Evaluation studies"
S6:"Achievement"
S7:"Psychology"
S8:"Judgement"
S9:"Problem solving"
S10:"Education, medical"
S11:"Education, nursing, baccalaureate"
S12:"Education, medical, undergraduate"

Figure 12. The neighbourhood of a document node (D0) in the supergraph. (See text).

the context graph contains some subset of the points in the supergraph, together with all the lines which connect those points in the supergraph. A change to the context graph can be specified simply by giving the set of points to be added to, or removed from it; the lines to be added or removed can be deduced.

(ii) unity. A truth value indicating whether the context graph is connected, or not.

(iii) explicit requests. This is a set $N_E \subset N$ of points either matching the user's expression of his interest (see section 3.2.4) or selected by him from displays.

(iv) inhibit list. A set $N_I \subset N$ of points explicitly or implicitly (by heuristics given below) rejected by the user. When points are being added to the context graph, those belonging to N_I are inhibited.

(v) last selected. A set of points $N_L \subset N$ selected by the user (sometimes implicitly) from the last display.

(vi) good documents. This is a set of points with document labels, $D_G \subset N$, which have been displayed to the user and elicited explicit approval from him.

(vii) accepted documents. A set of document points, $D_A \subset N$, which have been displayed, and about which the user has been non-committal.

(viii) reviewed nodes. There are occasions when the program chooses to display a node for the second time for the user's reconsideration. The set, N_R, of re-displayed nodes is maintained by the program.

(ix) performance. A number reflecting the history of the

user's reactions to the program's choices of what to show him.

At the beginning of a search, all the sets in the model are made empty. The following relationships between the sets are then maintained:

$$N_C \cap N_I = \emptyset$$

$$N_E \subset N_C \qquad \text{(and hence } N_E \cap N_I = \emptyset\text{)}$$

$$N_L \cap N_I = \emptyset$$

$$D_G \cap N_I = \emptyset$$

$$D_A \cap N_I = \emptyset$$

$$D_G \cap D_A = \emptyset$$

In other words, none of the points in the inhibit list are also in the context graph, all explicit requests are in the context graph, and the inhibit list, last selected, good documents and accepted documents are mutually disjoint sets of points.

3. Program function

The reader is reminded that this chapter is not concerned with implementation details, but rather to give a reasonably comprehensive understanding of the program's design. Decisions made during the design were made on the basis of such factors as the results and experience of others as reported in the literature, feasibility of effective implementation, and common sense (which still seems to have a significant role to play in this subject). Some of the features which govern the effectiveness of the system have been parameterized for convenient adjustment in experiments. The program was designed from the top, downwards, i.e. by progressive refinement, and this

description will follow the program structure through the top few levels.

There is very little to say about the top-most level of the program: it opens the disk files containing the data base and calls upon the topic search procedure as many times as the user requires. We move straight on to the topic search procedure (an Algol-like notation is used for the description of algorithms):

```
procedure TOPIC_SEARCH;
begin SET_UP_MODEL;
      repeat IMPROVE_MODEL
      until USER_SATISFIED
end.
```

At the beginning of each search, all the sets in the model are made empty by SET_UP_MODEL. The program is saying, in effect, "I know nothing about this user's interest". The structure of, and terminology used in the above procedure indicate the nature of the goals which the program tries to achieve - to improve its model, and thus, eventually, to get the user to express satisfaction. One might say that it is incidental to the main goal of IMPROVE_MODEL that it shows the user references to the literature. The user's reactions to those references are instrumental in improving the model. "IMPROVE_MODEL" is not always a very truthful label for the process it stands for, for a variety of reasons. If, for example, a user has seen all that the data base has on his interest, then either the model cannot be improved or, if it can, there is no purpose in doing so.

The sooner the user realises this and expresses "satisfaction", the better. In this case, alot depends on the user's confidence in the system, but there is a feature which prompts him (without compulsion) to stop the search.

Here, then, is the high-level definition of IMPROVE_MODEL:

```
procedure IMPROVE_MODEL;
begin message m;
      m:=GET_USER_MESSAGE;
      INFLUENCE_STATE_OF_MODEL(m);
      RESPOND_TO_USER(m)
end.
```

We describe the three processes invoked by IMPROVE_MODEL in the next three sections (3.1, 3.2 and 3.3).

3.1 The user's statement: GET USER MESSAGE

The function-procedure GET_USER_MESSAGE is of type message. Chapter 6 (sections 2.1, 2.2) explains the use of such type names for data structures in the development of the program. The value returned by the procedure is a representation of the user's statement, interpreted as a response to what the program last displayed. (The procedure is responsible for reading the statement). We must anticipate the section on RESPOND_TO_USER, and say what the components of a display are. Normally the program will display a reference using the label of a document node, followed by a numbered list of all the nodes adjacent to it. For example:

Misleading tests for glycosuria.; Feldman et al, Lancet,1,1246,2 Jun 73
1. J.M.Feldman, 2. F.L.Lebovitz, 3. false negative reactions, 4. glycosuria, 5. human, 6. methods

Sometimes the reference part of the display is absent; the display may be a collection of related subjects. Occasionally, there is neither reference nor numbered list (e.g. at the start of a dialogue).

The user's statement may be an instruction to stop the search, or it may give any of the following information:

(i) A relevance judgement on the reference shown (YES or NO),

(ii) An indication of what aspects he likes (or dislikes), using the numbers in the display,

(iii) One or more phrases or names related to his interest.

All parts of the statement are optional; in fact the user may make a null statement.

A message structure, m, produced by GET_USER_MESSAGE has four parts:

(i) reaction(m). This takes one of four values, which we shall denote STOP, YES, NO and NONE. If the value is STOP, the other three parts of m do not apply, otherwise it corresponds to the user's relevance judgement (NONE means that he did not give one).

(ii) select_list(m). A set of points which the user has explicitly or implicitly (see below) selected from the previous display.

(iii) reject_list(m). A set of points which the user has explicitly or implicitly rejected, from the previous display.

(iv) request_list(m). A list of items derived from the textual requests in the user's statement, structured for searching and matching with node labels in the supergraph. This is the only part of a message which has any meaning at the very beginning of a topic search.

The values of select_list(m) and reject_list(m) are derived from the last display, the user's statement and certain aspects of the model. The model is also modified. The actual algorithm is given below. We use the following symbols:

N_d is the set of points whose labels occur in the numbered list in the last display,

J = reaction(m),

$C \subset N_d$ is the set of points explicitly chosen by the user,

$R \subset N_d$ is the set of points explicitly rejected by the user,

N_E is the set of "explicit requests" in the model,

N_L is the set "last selected" in the model.

$\emptyset$, $\cup$ and - denote the empty set, the set union operator, and the asymmetric set difference operator, respectively.

The algorithm:

```
N_E := N_E ∪ C;
if C=∅ and R=∅ and J=YES then  C := N_d;
reject_list(m) := if R≠∅ then R
                  else if J=NO then N_d - (N_E ∪ N_L) else ∅;
select_list(m) := if C≠∅ then C ∪ N_L
                  else if J=YES then N_d - R else N_L;
N_L := C
```

It can be seen that certain assumptions are made about the user's intention. If he has given an unqualified YES (R and C both empty), it is assumed that he likes all the items displayed. It is assumed that he is still interested in the items which he chose last time (note that N_L is set at the end, ready for the next application of the algorithm). If his statement was an unqualified NO, the algorithm assumes that he would reject all the items displayed except those that he has chosen or explicitly requested earlier in the dialogue.

Another task performed during the interpretation of the user's statement is the categorization of the document node displayed according to the reaction part of the message. In the following algorithm,

J = reaction(m), having one of the values YES, NO or NONE,

d is the point whose document label has been displayed,

N_C is the set of points in the "context graph",

N_I is the "inhibit list",

D_G is the set of "good documents",

D_A is the set of "accepted documents":

```
case J of
begin
     NO: begin  N_I := N_I ∪ {d};
                N_C := N_C - {d};
                D_G := D_G - {d};
                D_A := D_A - {d}
         end;
     NONE: if d ∉ D_G then D_A := D_A ∪ {d};
```

/cont.

```
        YES: begin  D_G := D_G ∪ {d};
                    D_A := D_A - {d}
             end;
end
```

3.2 INFLUENCE STATE OF MODEL

The interpreted and structured statement is now used to modify the model as follows:

```
[Boolean stop_requested;]

procedure INFLUENCE_STATE_OF_MODEL(m);
message m;
if reaction(m)=STOP then stop_requested:= true
else begin
          COMPUTE_SCORE(reaction(m));
          PRUNE_CONTEXT(reject_list(m));
          ADD_TO_CONTEXT(select_list(m));
          FIND_NODES(request_list(m));
          UNIFY_CONTEXT_GRAPH
end.
```

We describe each of the five procedures invoked in turn.

3.2.1 Monitoring performance: COMPUTE SCORE

COMPUTE_SCORE is responsible for updating the numerical variable "performance" in the model to take account of the user's reaction to the last display. The value of performance is used by RESPOND_TO_USER, under certain circumstances, to determine what should be displayed next,

which in turn influences the future states of the model. Hence, the method of calculating "performance" influences the program's effectiveness. We want a measure of the program's success which "remembers" past performance, but gives greater weight to the recent past. A simple formula is used, which computes the (n+1)th performance, p_{n+1}, from the nth value, p_n, and the success rating of the last interaction with the user, X_J:

$$p_{n+1} = Mp_n + X_J$$

M is a constant, the "memory factor", and should have a value in the range $0 \leqslant M \leqslant 1$. The value of X_J depends upon the reaction, J, passed to COMPUTE_SCORE. One set of values which has been used is

$$M = \tfrac{1}{2},\ X_{NO} = -1,\ X_{NONE} = 0,\ X_{YES} = +1,\ p_0 = 0.$$

3.2.2 Removing points from the context graph: PRUNE_CONTEXT

The procedure PRUNE_CONTEXT deals with the points which the user is assumed not to like in the last display, i.e. reject_list(m) in message m. As usual,

N_C is the set of points in the "context graph",

N_I is the "inhibit list", and

N_E is the set of "explicit requests".

```
procedure PRUNE_CONTEXT(rejects);
point set rejects;
begin  N_C:= N_C- rejects;
       N_I:= N_I∪ rejects;
       N_E:= N_E- rejects
end.
```

Notes: (i) Removal of points from the context graph implies removal of lines incident with them.

(ii) It is possible to remove points from "explicit requests". Thus a user can change his mind about what subjects he is interested in.

3.2.3 Adding points to the context graph: ADD TO CONTEXT

The procedure given below adds points to the context graph. It also brings into the context graph document points adjacent, in the supergraph, to the new points.

```
procedure ADD_TO_CONTEXT(chosen);
point set chosen;
begin  N_I := N_I - chosen;
       N_C := N_C ∪ chosen;
       N_C := N_C ∪ LINKED_DOCUMENTS(chosen)
end.
```

To define the set that LINKED_DOCUMENTS produces, we first recall some notation from section 1 of this chapter. The supergraph, $S = (N,L,A)$, where N is a set of points, L the set of their labels and A is a set of lines. The set of points adjacent to a point $n_i \in N$ is

$$s_i = \{m: \{n_i, m\} \in A\}.$$

Let $C = \text{chosen} - N_{Ch}$, where N_{Ch} is the set of "check tags"*. C is a subset of N, say $\{n_{k_1}, n_{k_2}, \ldots n_{k_q}\}$. The set

---

* Check tags are subject points which are adjacent to relatively many document points. They correspond to terms with high postings in MeSH. See section 3.1.1, Chapter 3, and section 2, Chapter 7.

returned by LINKED_DOCUMENTS consists of all the members of the set

$$\bigcup_{1 \leq i \leq q} s_{k_i} - N_I$$

whose labels are of type "document".

3.2.4 Incorporating textual requests: FIND NODES

The task of matching a word, phrase or name suggested by the user with a node label in the supergraph is fairly complicated in this program. It is more than simple string matching. A description of the techniques used will be found in Chapter 5. Here, we concentrate on the effect upon the model of such initiative by the user. In the expression of FIND_NODES that follows, we use the usual notation for components of the model, namely N_E for the "explicit requests" set, N_I for the "inhibit list" and N_C for the set of points in the "context graph". The process denoted by LOCATE_NODES produces the set of points matching the requests. This set may be empty, or it may contain more than one match for some of the requests. The function STARS occurring in FIND_NODES, below, is very similar to LINKED_DOCUMENTS (see section 3.2.3), but there is no restriction as to the type of points that are included in the result.

```
procedure FIND_NODES(requests);
query list requests;
begin point set P;
      P:= LOCATE_NODES(requests);
      N_E:= N_E ∪ P;
```

```
      N_I := N_I - P;
      N_C := (N_C ∪ P) ∪ STARS(P)
end.
```

Not only are the located points included in the context graph, but also all the non-inhibited, non-"check tag" points adjacent to them in the supergraph.

3.2.5 Establishing coherence: UNIFY_CONTEXT_GRAPH

When the context graph has been modified, points added and removed, UNIFY_CONTEXT_GRAPH is executed to find out if the context graph is connected (i.e. in one piece), and if not to attempt to join the separate components by adding a few appropriate points from the supergraph. If, when it is done, the context graph is connected, the Boolean variable "unity" in the model will be *true*, otherwise it will be *false*. In the procedure, G_C is the context graph and $|K|$ denotes the number of elements in the set K.

```
procedure UNIFY_CONTEXT_GRAPH;
begin graph set K;
      K := CONNECTED_COMPONENTS(G_C);
      if |K|≤1 then unity:=true
      else
      begin  DISCARD_USELESS_COMPONENTS(K);
             unity:= if |K|>1 then TRY_JOIN(K) else true
      end
end.
```

The procedure CONNECTED_COMPONENTS finds all the maximal connected subgraphs of its argument, G_C. It does this by picking any point, p, in G_C and locating all the points, also in G_C, reachable from p. Those points together with p form the first component. If there are any points in G_C which were not visited in the search, one of them is chosen and the process is repeated to produce the next component; and so on, until all the points in G_C have been used. The result in general is a set of graphs with mutually disjoint sets of points. If this set has more than one member, we should like to find paths in the supergraph which join them together. However, implementation must be considered at this point. We could use a technique very like that used to determine the connected components of the context graph. Think of the points in a component as a wavefront. Now advance the wavefront by moving along each line which connects a known point to an unvisited one in the supergraph. To find a path between two components, advance the two "wavefronts" alternately until they meet at some point. Backward links must be recorded everywhere throughout the process, so that the path can be determined from the meeting point. (Quillian,1968 implemented this method in his semantic memory). The process is rather expensive, and there is a user waiting for a response. Unlike the context graph, which is stored in fast storage (virtual memory in our implementation), the supergraph sprawls across magnetic disk, and logically adjacent nodes will often be widely separated in storage.

The procedure given above first tries to reduce the problem by invoking DISCARD_USELESS_COMPONENTS. Some

critical points (cutpoints) may have been removed from the context graph, isolating small components. If a small component has no points which are members of N_E (explicit requests) or N_L (last selected), it is deleted from the context graph, and from the set R. We can adjust the meaning of "small component": it might mean components with less than 3 points, for example. These deletions may have reduced the context graph to a single connected component, but if that is not the case a quick attempt is made to join them by TRY_JOIN. If it does not succeed, it returns the value false and "unity" (in the model) remains false, therefore.

Going back to the wavefront analogy, each component/wavefront is advanced one step (from all points in the component except "check tags" to non-inhibited points in the supergraph). The new "wavefronts" are intersected in pairs and single points are chosen from the non-empty intersections, preference being given to document points. These points are added to the context graph using ADD_TO_CONTEXT (see section 3.2.3). TRY_JOIN never advances the "wavefronts" more than one step, so the backward chaining referred to above is not needed.

This completes the description of the process named INFLUENCE_STATE_OF_MODEL.

3.3 RESPOND TO USER

Now that the user's statement has been used to modify the model, a suitable response is determined by the program from the model. The program aims to give the user pertinent references. In order to do this it must collect

suitable information from the user. Sometimes it is better to make a provocative response than to give the "best" reference from a dubious model.

In this procedure, N_C denotes the set of points in the context graph:

```
[Boolean stop_requested;]

procedure RESPOND_TO_USER(m);
message m;
begin point d;
      if not stop_requested then
      begin if N_C=Ø then STIMULATE_USER
            else
            if reaction(m)=YES then
            begin if last display contained a reference, d
                        then DISPLAY_SIMILAR(d)
                        else PICK_A_DOCUMENT
            end
            else
            if performance is low then REVIEW_COURSE
            else PICK_A_DOCUMENT
      end
end.
```

In sections 3.3.1 - 3.3.3 we discuss PICK_A_DOCUMENT, DISPLAY_SIMILAR and REVIEW_COURSE. STIMULATE_USER is a simple procedure which tries to reintroduce references or topics in which the user has previously shown interest.

3.3.1 Using the context: PICK A DOCUMENT

This procedure for determining what to show the user is actually invoked more often than the definition of RESPOND_TO_USER would suggest, because under certain circumstances, DISPLAY_SIMILAR also calls upon it. It is the procedure which assumes that the context graph is a reasonable representation of the area of the user's interest, and therefore tries to make a sensible choice from the document nodes contained in it.

In the definition of the procedure, N_C is the set of points in the context graph G_C, and unity is the truth valued part of the model which indicates whether G_C is connected.

```
procedure PICK_A_DOCUMENT;
begin point set D;
      D:= UNSEEN_DOCUMENTS(N_C);
      if D=∅ then SUGGEST_SUBJECTS
      else DISPLAY_DOCUMENT(if unity then MOST_INVOLVED(D)
                                     else AVERAGE_INVOLVED(D))
end.
```

UNSEEN_DOCUMENTS(N_C) produces those members of the set $N_C - (D_G \cup D_A)$ which have document type labels. (D_G and D_A are the sets "good documents" and "accepted documents", respectively. Documents which have been seen and rejected will be represented in the "inhibit list", N_I. We can forget them because $N_I \cap N_C = \emptyset$). SUGGEST_SUBJECTS displays a collection of subjects related to one of the user's explicit requests (see section 3.3.3 in this

chapter). The form of display produced by DISPLAY_DOCUMENT has already been described (section 3.1). We come to the concept of involvement in the context graph, in order to elaborate MOST_INVOLVED and AVERAGE_INVOLVED. The connect coefficient of a point, p, in the context graph, G_C, is defined to be:

$$\frac{\text{degree of p in } G_C}{\text{degree of p in the supergraph}} .$$

The degree of a point in a graph is the number of lines in the graph which are incident with the point. The values taken by connect coefficients range from zero, for an isolated point, to 1 for a point all of whose immediate neighbours in the supergraph are also in the context graph. We use the connect coefficient to measure the involvement of points in the model. MOST_INVOLVED finds the member of its argument which has the highest connect coefficient. AVERAGE_INVOLVED finds the point with connect coefficient closest to the average of the coefficients of all the members of its argument. It is used when the last attempt to join up the components of the context graph failed, and can be regarded as the next heuristic in the effort to form a connected context graph. By giving the user something near the periphery (but not so near that he rejects it out of hand), we hope for guidance on how to extend the context graph: TRY_JOIN might succeed next time.

3.3.2 DISPLAY SIMILAR

The user has approved of the last reference that was displayed. Now the program will try to find a document node "like" it, regardless of the context graph; i.e. it

will be prepared to look anywhere in the supergraph. Similarity measures between documents indexed by keywords have received much attention in the literature, and a discussion of the topic in relation to our program will be found in Chapter 3, section 3.1.1. Similarity between documents is usually taken to mean similarity between their sets of index terms. Typically, if two documents have keyword sets X and Y respectively, the extent of their similarity to each other would be given by

$$\frac{|X \cap Y|}{\text{Normalizing factor}}.$$

The normalizing factor is a number which takes into account the sizes of X and Y, e.g. $|X| + |Y|$.

An equivalent measure in our system would be based on the sets of points adjacent, in the supergraph, to the two points whose similarity is to be measured. In fact, the measure used also takes into account the user's expressed interest and, in a primitive way, the usefulness of the subject terms as distinguishers between documents.

We now define the similarity measure between two points d_1 and d_2 in the supergraph $S = (N,L,A)$.
Firstly, $d_1 \in N$ and $d_2 \in N$.
Now let I_1 be the set of points adjacent to d_1, and I_2 be the set adjacent to d_2, i.e.

$$I_1 = \{n: \{n,d_1\} \in A\}$$
$$I_2 = \{n: \{n,d_2\} \in A\}$$

Let $E = I_1 \cap N_E$, where N_E is the "explicit requests" set in the model.
Let $T = I_1 - (N_I \cup N_{Ch} \cup E)$, where N_I is the "inhibit list"

in the model, and N_{Ch} is the set of "check tags" (which are regarded as not very useful for this purpose).
The similarity function is

$$sim(d_1,d_2) = \frac{\alpha |E \cap I_2| + \beta |T \cap I_2|}{|I_2|}$$

where α and β are adjustable constants, which determine the relative importance given to explicit requests. The numerator is actually symmetrical with respect to d_1 and d_2; it is just expressed in a form that corresponds quite closely to the way in which the program works it out. As a whole, however, the function is not symmetrical because the denominator (normalizing factor) is not.

To define the action of DISPLAY_SIMILAR, we shall use the same notation as used above. The meaning of UNSEEN_DOCUMENTS is as given in section 3.3.1 above. τ is another adjustable constant.

```
procedure DISPLAY_SIMILAR(d1);
point d1;
begin point d,di;  point set D;
      D:= UNSEEN_DOCUMENTS(I1);
      if D≠∅ then DISPLAY_DOCUMENT(any d∈ D)
      else
      begin find di ∈ UNSEEN_DOCUMENTS(N) for which sim(d1,di)
            is maximum;
            if sim(d1,di)≥τ then DISPLAY_DOCUMENT(di)
            else PICK_A_DOCUMENT
      end
end.
```

The procedure first looks for documents directly related to the parameter, d_1. If it finds any it picks one for display, otherwise it finds the document most similar to d_1 and displays that, unless it is not similar enough, in which case a document is chosen from the context graph. τ, the "similarity threshold", is used to determine whether the most similar document is similar enough.

3.3.3 REVIEW COURSE

We shall now deal with the action taken by the program when its performance falls too low. The overall strategy is as follows:

(i) Look for a reference which the user has already seen and not rejected, and display it again, asking him to reconsider it.

(ii) If the search for a suitable document point fails, show the user one of his explicit requests together with its adjacent subject nodes.

(iii) If no such point can be found, ask the user to take the initiative and think of a new term or name.

In the procedures that follow,

D_G is the set of "good documents",

D_A is the set of "accepted documents",

N_R is the set of "reviewed nodes", and

N_E is the set of "explicit requests", all in the model.

```
procedure REVIEW_COURSE;
begin point set D;
      ADMIT_FAILURE;
      D:= D_G - N_R;
```

```
      if D≠Ø then RE-DISPLAY (LEAST_INVOLVED(D))
      else
      begin D:= D_A - N_R;
            if D≠Ø then RE-DISPLAY (MOST_INVOLVED(D))
            else SUGGEST_SUBJECTS
      end
end.
```

ADMIT_FAILURE confesses failure to the user; it will, however, point out that he may have seen enough if he has approved of a few of the references shown him. The set N_R is used to ensure that nothing is reviewed more than once. RE-DISPLAY and DISPLAY_SUBJECTS (called by SUGGEST_SUBJECTS, below) each add their argument to N_R. If there are "good documents" to review, we assume that sometime during the dialogue, the context graph has been allowed to "grow" in the wrong direction. Therefore, we should give the user maximum opportunity to indicate new directions: hence the use of LEAST_INVOLVED when $D_G - N_R$ is not empty.

```
procedure SUGGEST_SUBJECTS;
begin point set E;
      E:= N_E - N_R;
      if E≠Ø then DISPLAY_SUBJECTS (LEAST_INVOLVED(E))
      else tell the user to give a new term or name
end.
```

DISPLAY_SUBJECTS produces a numbered list of subjects for the user to inspect. The points chosen for the display are the argument of DISPLAY_SUBJECTS (if it is a subject point)

and all the subject points adjacent to it. A sample display (the argument of DISPLAY_SUBJECTS has the label "antibodies"):

1. antibodies, 2.anti-antibodies, 3. autoantibodies,
4. binding sites, antibody, 5. immune serums,
6. insulin antibodies, 7. immunoglobulins,
8. isoantibodies, 9. plant agglutinins

The user can respond to this with the type of statement outlined in section 3.1, which will be read and interpreted by GET_USER_MESSAGE. He may even give a general judgement (YES or NO) which will be used by the program in the usual way, except where the last reference displayed would normally be processed.

3.4 Other features of the program

In a full-scale operational system the interface with the user would have to be very much more sophisticated than in our prototype. We have, however, made three small concessions to human engineering - the "slate", provision of help, and automatic printing of hard copy.

Conceptually, the slate is a separate display of limited capacity, independent of the one used for the main dialogue. For the present, rather than link two real screens, the independence is simulated using one screen, and the user can switch to the slate, manipulate it and switch back to the first "screen", at any time. Items (references, names, subjects) that crop up in the main dialogue can be recorded on the slate purely for the user's convenience, and no inferences are made by the program about his area of interest.

Help can be obtained from the program by typing a question mark (?). A display appropriate to the area of

dialogue that is being conducted will be shown. The user presses a button when he is ready to go on.

At the end of each topic search, the contents of the slate, and the document labels of all the points in "good documents" and "accepted documents" are sent to the line printer.

The above features are for the user's benefit. There are two more capabilities which are present for experimental purposes - conversation logging and a model-snapshot routine. All dialogues with the program are copied to the printer for later inspection. At any stage in a search a request can be made to take a snapshot of the model. A numerical representation of the current state of the model is quickly copied to a file, and the dialogue can continue. There will be an indication in the log at the point where a snapshot has been taken.

4. Summary

We have given, in this chapter, an abstract and fairly detailed description of the bibliographic retrieval system. The important aspects of the program have been described, but large and complex pieces of program have been glossed over - particularly matters of file organization and searching - because they are not central to the topic of this thesis. Also, we have said very little about implementation of the processes described - either about algorithms or about programming methodology. There have only been scant hints of justification for the way the program is. All these matters are dealt with in other chapters (3, 5 and 6). What we have given is a "reference

manual" from which some properties of the program can be deduced. The set- and graph-theoretic notations and terminologies are those of Halmos(1960) and Harary(1969), respectively.

Chapter 5

DATA RECOGNITION AND FILE ORGANIZATION

In most information systems, the enquirer must take the initiative at least once and indicate his area of interest. In our system, he can do this as little or as often as he wishes, and the effect that his actions have is described in Chapter 4. We start now by concentrating on the way in which textual information (titles, personal names, subject terms or phrases) typed by the user are transformed into sets of points in the "supergraph" for use in maintaining the program's "model" (see Chapter 4, section 3.2.4).

A statement made by the user at the terminal may contain several separate pieces of text: they are dealt with, one after the other, by the program, which constructs the union of the sets of points whose "labels" (Chapter 4, section 1.1) match them. We shall limit our consideration here to the means of matching just one textual request. Even so, the result may not be simply a single point; there may be several or, of course, none at all.

Two features are required of a text (or string) matching mechanism in the circumstances of an on-line search. Firstly, it should be helpful; that is it should accommodate inaccuracies and variants to some extent, so that it does not turn away a user who cannot supply, for example, a complete name or a title in exactly the right form. Secondly, it should work speedily, and this naturally places limits on how helpful we can make the program in this respect. There are two reasons, however, why we need

not use all the sophistication of the modern computational linguist in this problem. The first is the so called "law of diminishing returns": we can get quite good algorithms quite easily, but however large and complex programs become, there is always yet another special case to deal with. The second reason stems from the nature of the problems involved in dealing with the more distant variants (synonyms, for instance): we are not tackling problems in information retrieval by vocabulary control or manipulation, but by a new form of dialogue and representation of the searcher's interest. However, the problem of inexact string matching is an important aspect of systems design for non-delegated searches, so we have given it more than passing attention. The details of the techniques used are given in this chapter. The file structure supporting these techniques and the supergraph will also be discussed.

1. Matching user's requests in the data base

Text input from the user's terminal is considered by the program to be a "stab in the dark", in the sense that the enquirer is not expected to know the exact form of the names and phrases stored in the system, or to use a thesaurus. Common reasons for mismatch between what he types and what is stored are inaccurate spelling, particularly of names, defective memory of long titles, variant word order or grammatical form in subject terms.

With the exception of a very few systems which perform complex linguistic analysis of queries (e.g. LEADERMART - Hillman,1973), on-line reference retrieval systems tend to use exact string matching. The help given

to a user who is not sure of the "vocabulary" depends very much on the file organization already chosen to facilitate some other aspect of system design. For example, the Retrospec I system (see Goodliffe & Hayler, 1974), which uses the "Computers and Control" part of the INSPEC (Information Services in Physics, Electrotechnology, Computers and Control) data base, accepts combinations of character strings (written between quotes) such as

'STRING' AND 'MATCHING',

and scans sections of the file (as requested by the user) for records which contain the specified combination in the title or index term fields. The onus is entirely upon the user to formulate a query which will not miss variants, such as 'BEST-MATCH SUBSTRING SEARCHING'. Of course, more complicated matching is possible (in fact, a simple term weighting scheme is implemented in Retrospec I), but with large files the sequential search method imposes its own limitations in the on-line situation.

A file organization which is particularly effective for one type of access is often inhospitable to others. It is, for example, difficult to do inexact matching in a large ordered index, or a list-based structure. Pre-processing index entries (e.g. stripping word affixes) and identically preprocessing queries can be effective. Alternatively, predictable variant spellings, and even synonyms, can be included in the vocabulary with references to the "correct" terms (e.g. Medlars - Barraclough,1972; and the European Nuclear Documentation Service vocabulary is reported - Vernimb & Steven,1973 - to contain some 60,000 previously detected erroneous spellings). If the

entry vocabulary is contained in an ordered index (designed, perhaps, for binary searching), it is not difficult to give the user the ability to scan alphabetical neighbours. In the Medusa system (see Chapter 2, section 2.4.1), for example, the command

```
LIST DIABET
```

will cause all terms beginning with the characters 'DIABET' to be displayed:

```
DIABETES BRONZE
DIABETES FRAGILE
    .
    .
DIABETIC ACIDOSIS
    .
    .
DIABETIC RETINOPATHY
```

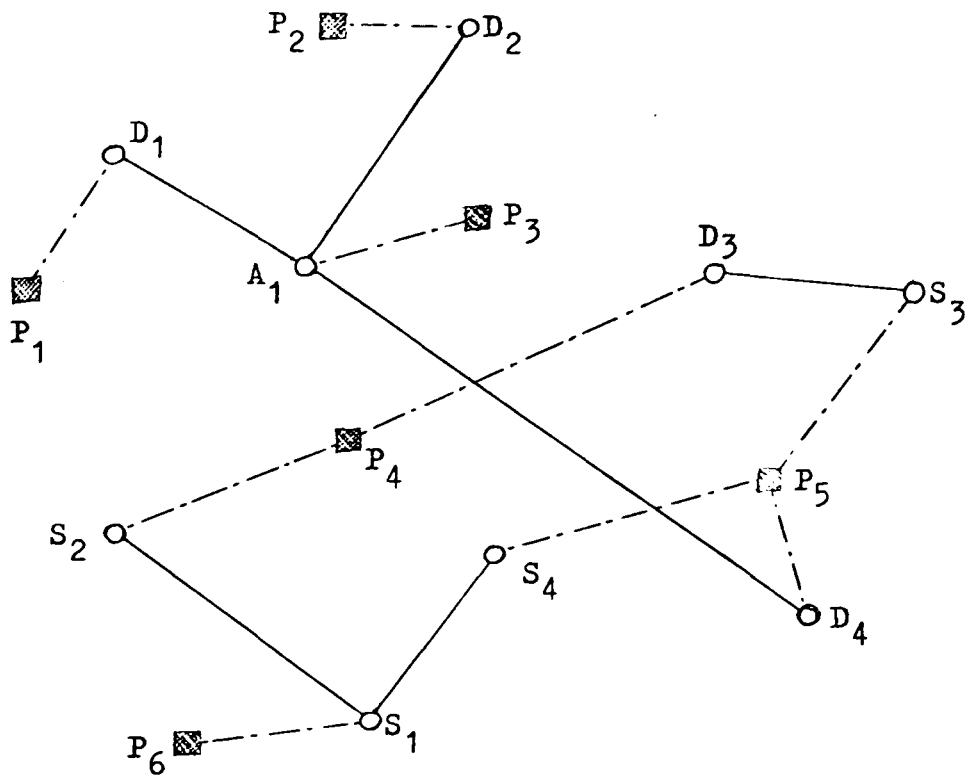
(It is a facility which is rarely used in practice). Rickman & Walden(1973) have described an interesting (and efficient) file structure for on-line thesaurus searching, but even there no attempt is made to make inexact matches.

Variations on these themes are numerous and we shall not cover them exhaustively here. The one further class of techniques which we should mention is that of correcting misspellings by measuring the similarity of an object word with each member of a vocabulary and picking the most similar (Alberga 1967, Blair 1960, Morgan 1970). The prime motivation for this work has been to produce operating systems and compilers which are reasonably insensitive to spelling errors (Wagner,1974). These techniques are, however, unsuitable for very large vocabularies, and although Szanser(1973) has tackled the size problem, it is doubtful that this approach would be very productive in the bibliographic search environment in view of the nature of the more troublesome inaccuracies.

1.1 Partitioning the bibliographic labels

Now we come to the technique used for text matching in the present system. As records ("labels") are added to the data base ("supergraph"), they are organized into disjoint partitions according to certain lexical features. This organization is overlayed upon the supergraph, but is independent of it - two members of a partition may or may not be neighbours in the supergraph. One way of visualizing the whole structure is depicted in figure 13. The partitions have names, or codes, denoted in the figure by P_1, P_2, ... , which are derived from labels by the compression algorithms described in the next few pages. These algorithms are designed to produce a single code for many of the variants of a piece of text. The solid, square nodes, in the figure, act as the "centres" of partitions. Each circular (supergraph) node is attached, by a broken line, to exactly one square node; the partition named P_i is the set of points adjacent to the point labelled P_i. When a new point's label is compressed, the partition bearing that name is sought. If it is found, the new point is added to it (by drawing a new broken line, in the pictorial analogy), otherwise a new partition is created (in the picture, a new square joined to the new circle).

Incoming textual requests are processed by the same algorithms as handled the labels before them, and a partition is thus identified and searched for a best match. There is a resemblance between this method and conventional scatter storage of records in multiple entry buckets (Buchholz,1963). However, whereas most "randomizing" functions will place otherwise unrelated records in the



Key (i) author labels: A_i

(ii) document labels: D_i

(iii) subject labels: S_i

(iv) partition names: P_i

Notes (i) The partition named P_i is the set of points adjacent to the point labelled P_i (i.e. joined to it by broken lines).

(ii) If the solid, square nodes and the broken lines (incident with them, without exception) are deleted the supergraph remains.

Figure 13. Partitions overlayed on the supergraph.

same bucket, we require one which collects together labels which look similar. Another point of difference is that our partitions are not fixed capacity "stores", but arbitrarily sized sets of records. Once a point has been located in response to a textual request, the partitions can be forgotten - the square nodes and the broken lines in figure 13 can be ignored - for they are not used in subsequent supergraph manipulations.

In connection with on-line searching of a library catalogue, Kilgour and his associates have experimented with simple truncation of title words with a view to partitioning the catalogue (Kilgour 1970, Long 1972). Leading non-significant words are removed and subsequent words truncated to lengths specified in a vector. For example, a truncation function based on the vector (3,1,1,1) creates keys (or partition names) comprising three letters from the first word and the first letter of each of the next three words. The partitions are small (size is roughly hyperbolically distributed; typically 99% of partitions have less than 10 members in a collection of 100,000 titles), and spelling mistakes have little effect on searching. On the other hand, word order errors cause havok, and in general there is not much orthographic similarity within the partitions.

It was decided to treat proper names differently from phrases (titles and subject terms) in the present program, because the types of error people make are different in the two classes of data. Whichever of the two algorithms is used, the result is a four-character code, and this, together with an indication of the type of the original

data (name or phrase), is the name of the partition to which the label should belong, or which should be searched for a good match in the case of a query.

1.1.1 Proper name compression

The most famous name compression algorithm is SOUNDEX (Wright,1960). Its aim is to compress names into short codes so that those with similar sounds have identical codes. More recently, an algorithm which outperforms it was devised by Dolby(1970), and it is the one that we use here, with minor modification. Nugent(1968) has produced a review of several methods, but not all will generate partitions useful for our purpose. Dolby applied his method to the names in a telephone directory and then compared the equivalence classes obtained with those given, manually, by the compilers of the directory, in the form of *see also* cross references. The method correctly provided 80% of the man-assigned classes and improperly split only 5·3%. The same experiment using SOUNDEX resulted in corresponding proportions of 63·8% and 30%. As Dolby points out, these figures are not a direct gauge of performance with erroneous names, but he beleives that they provide a good indication, and this is substantiated by the observations of Tagliacozzo *et al*(1970). Seventy-seven errors collected during a survey were analyzed in some detail and the letters involved in the errors listed. The full context of the errors are not given in the paper, but one can deduce that 52 (67.5%) of the errors would definitely not have affected the code produced by Dolby's algorithm. Of the remaining 25, some would very likely also have been

inconsequential.

Forenames and initials are not used and the order of execution of certain steps of the algorithm, which follows, is important.

(i) Leading Mcg, Mc, Mac or Mag is replaced by Mk.

(ii) The second letter in each occurrence of dt, ld, nd, nt, rc, rd, rt, sc, sk, st is removed. This is done working from the right hand end of the name, recursively.

Note: the sound of the deleted letter is usually indistinct in these contexts.

(iii) The following replacements are performed throughout the name:

x by ks, ce by se, ci by si, cy by sy, ch by sh when preceded by a consonant, any other c by k, z by s, wr by r, dg by g, qu by k, q by k, t by d, ph by f.

(iv) If a consonant, excluding l, n, r, occurs after the first position in the name and immediately before k, it is removed.

(v) One letter is removed from every doubled consonant.

(vi) pf at the end of the name is replaced by p; pf at the beginning is replaced by f.

(vii) gh at the end of the name is replaced by f if preceded by a vowel, or by g otherwise; gh anywhere else is deleted.

(viii) The first two vowel strings are replaced by a vowel string marker (a single character, represented by the letter a, which has become free by virtue of this step); subsequent vowel strings are removed. For

this purpose, a vowel is one of the letters a, e, i, o, u, y and (in all but the first position) w and h.

(ix) The four-character code is obtained: if the name now has less than 4 characters it is padded with blanks on the right; otherwise, the name is truncated to 6 characters and, as long as there are more than 4 characters, vowel string markers are removed, starting with the right-most one. Finally, the name is truncated to 4 characters if necessary.

The following letters cannot occur in the compressions of names: x, c, z, q, t, e, i, o, u, y. In addition, w and h can only occur in the first character position, and the blank may not occur there. So one can have at most $16 \times 15 \times 15 \times 15 = 54{,}000$ partitions of proper names, which is adequate for collections of order 100,000 documents. For larger collections the code might be increased in length by simply modifying step (ix), above. Five characters could generate 810,000 partitions, for instance. Table 1 shows some examples of partitions and erroneous names which would identify them.

Table 1. Partitions of proper names.

partition	matching names	code
Nilsson B.S. Nilson K.	Nelson Nillson	NLSN
Muller M. Muller W. Muller H. Mollard P. Miller S.A.	Mahler Mueller Mallory	MALR
Stieglitz P. Sziegoleit W.	Siegleitz	SGLD

/cont.

Kuhn R.A. Kahn K. Cohen J.G. Cohen D. Cohen G.	Cohn Kant	KAN[illegible]

Table 1. (concluded)

1.1.2 Phrase compression

Ayres et al(1968) found that, in the special research library environment, titles are given remarkably accurately in requests and that most errors either occur after the first few words or consist of word inversion or the omission of commonplace words such as 'report' or 'outline'. We treat subject requests and titles identically and, where appropriate, a partition may contain both document and subject labels. This means that what was meant by the user simply as a subject descriptor may best match the title of a document; and that is beneficial to the operation of the system. Ayres' results do not necessarily apply to subject phrases, which tend to be invented rather than recalled by searchers. However, techniques based on his observations work reasonably well for subjects mainly because the phrases are generally short. We can reduce the effect of errors or variations in phrases quite simply by removing non-significant words and suffixes, leaving a sequence of presumably meaningful stems. The reliability of the stems decreases as we go along the sequence, so we only use the first two. Word order variation is coped with by applying a symmetric function to the two stems (if there are as many as two, of course). In describing the phrase

compression algorithm, we shall make use of two examples from the medical test collection:

E1: Urbanization and mental health: a reformulation

E2: Apropos of the article: "Systemic venous insufficiency. A new and rare syndrome"

The first step is to select two "significant" words from the phrase, scanning from the left. A dictionary of common words is used for this task. The stop list published in the Science Citation Index was modified to suit the subject matter of the collection. 657 words appear in the dictionary and are of two types: words which are always discarded from the phrase (145 of these), and words which are only used if there are insufficient significant words (i.e. words not in the dictionary). Table 2 shows some of the dictionary. The words selected from our examples are:

E1: Urbanization, mental

E2: Systemic, venous

Table 2. Sample from the dictionary of common words

These words are always ignored	about, against, and, best, but, concerning, easy, few, given, have, instead, look, make, next, other, same, several, that, the, when, with (also all one-character words)	Total in dictionary 145 words
These words are only used when significant words are scarce	addendum, affect, apropos, assumptions, body, cell, characteristic, clinical, conference, definition, device, erratum, evaluation, gram, implication, important, introduction, measure, medical, optimal, organ, proceedings, quality, standard, theoretical, volume	Total in dictionary 512 words

Each selected word is then stripped of its suffixes. Resnikoff & Dolby(1965 & 1966) have produced two very useful analyses of English affixes and their lists, slightly modified to suit the subject matter, are used. The lists for long and short words (measured in this program by counting vowel strings) are not identical and are given in Table 3. The largest suffix that can be identified is removed, and the process is repeated on the remainder of the word until it has no identifiable suffix. Let us apply this to the examples:

Urbanization → Urbaniz → Urban → Urb

mental → ment

Systemic → System

venous → veno

E1: Urb, ment

E2: System, veno

The "stems" are then abbreviated to four characters, in such a way as to preserve discrimination between different word fragments as much as possible. Bourne & Ford(1961) give several techniques and one has been chosen which, in their experiment, retained discrimination for 98·2% of their vocabulary of 2082 words. Starting with the second letter, every alternate letter is dropped until only three letters remain. If there are still more than three letters when the end of the word is reached, the process is repeated. All the dropped letters are "added" together, modulo 27, to produce the fourth character of the abbreviation.

E1: Urb␣, mnte

E2: Ssed, vnoe

Short-word suffixes (to be removed from 2-vowel-string words)

-a	-ure	-al	-o	-let
-ic	-ite	-el	-ar	-et
-ed	-ue	-ful	-ier	-ant
-land	-ive	-um	-ler	-ment
-ward	-e	-man	-er	-ent
-ard	-ling	-an	-or	-ot
-ee	-ing	-en	-is	-ow
-age	-ah	-in	-less	-ey
-ie	-ish	-eon	-ness	-ly
-ile	-lock	-ion	-us	-y
-ine	-ock	-on	-at	-iz

Long-word suffixes (to be removed from words with more than 2 vowel-strings)

-ia	-ine	-i	-ation	-at
-oma	-ure	-ical	-ion	-et
-a	-ise	-eal	-on	-it
-ic	-ose	-ial	-o	-ant
-ed	-ate	-al	-ar	-ient
-oid	-ite	-el	-ular	-ment
-ance	-ette	-ol	-eer	-ent
-ence	-yte	-ful	-er	-est
-ide	-ue	-ism	-or	-ist
-ee	-ive	-ium	-is	-ly
-age	-ize	-um	-ess	-ary
-ie	-e	-ian	-eous	-ery
-able	-ing	-an	-ious	-ry
-ible	-og	-gen	-ous	-y
-ile	-ish	-in	-us	-iz

The letter s is removed from any complete word from which no suffix can be removed.

Table 3. Suffix lists for phrase compression.

Finally, regarding the letters in the codes as digits in the base 27 number system, add the two codes, modulo 27^4 to obtain one four-character code. This is a symmetric operation, as required.

E1: HEVE

E2: OFTI

The number of different phrase partitions which can be named is 27^4 = 531,441. In conclusion, this method of phrase-compression maintains discrimination between the expected information-bearing parts, namely the stems of significant words, and equates phrases which differ only in the less memorable parts - non-significant words and suffixes. Examples of partitions obtained are given in Table 4.

Table 4. Partitions of phrases.

partition	matching phrases	code
Urbanization and mental health: a reformulation (document)	Effect of urbanization on mental health	HEVE
attitude of health personnel attitude to health	attitudes to health personnel healthy attitudes	IUYN
Does hemorrhagic shock damage the lung? (document) shock, hemorrhagic	hemorrhagic shock	␣WS␣
alkaloids alkalinity alkeran	alkaline	ALK␣

1.2 The matching process

To summarize the contents of the preceding paragraphs, the texts of the labels in the system are compressed to form codes which generate a partitioning of the corresponding points in the supergraph. The matching process consists of similarly compressing user's text to identify a partition, and then finding a satisfactory point within it. The questions that arise are: which compression

procedure should be used? what should be done if the search fails?

If a user has enclosed the string in quotation marks, it is assumed to be a phrase, otherwise the initial assumption is that it is a name and the program isolates the surname and forms a string of initials if suitable data is present. The appropriate compression is performed and a search made for a partition with the derived identification. If there is no such partition, a sequence of automatic re-tries are made:

(i) If the string was assumed to be a proper name, then it is re-interpreted as a phrase; the user may have forgotten the quotes.

(ii) If two words were used to form a phrase compression code, they are tried singly, the first in the phrase, and then the second if necessary.

(iii) Failing the automatic tries, the program invites the user to replace that part of his statement. If he wishes he can simply have that part ignored.

The main disadvantage of the method is that a one-word query cannot match a two-word label: for example, 'carotid' will not retrieve the partition containing 'carotid arteries'. In the reverse situation, this may happen: 'membrane antigens' retrieves no partitions, however 'membrane' does, and the search stops there. It might be better to look for 'antigens' so that both aspects of the topic are represented. As the program stands, the user would be consulted about the acceptability of 'membrane', and can explicitly introduce 'antigens' if he feels the need.

When a partition is found, the labels are accessed and ranked according to similarity with the request (the measure of similarity is very simple and we shall not go into it here). If there is one outstanding match, the corresponding point is selected without troubling the user, and the matching process is complete. If the choice is not obvious, the user is asked to make the decision: he can accept as many of the displayed labels as he likes, including none at all. In the latter case the program behaves just as though no partition had been found and goes on to the next "automatic re-try".

Examples:

(i) User's text: artificial respiration

System action: No quotes, so tries to find name A.Respiration. No partition found, so re-interprets text as a phrase and finds a partition containing respiration, artificial. Match is good enough, so the corresponding point is selected.

(ii) User's text: Millen

System action: Partition found containing the names

D.Moulin

J.Milin

R.Milin.

No outstanding match, so user is shown all three and asked to choose.

User: Not happy with any of them, rejects all.

System action: Tries 'millen' as phrase without success and invites user to try a

substitute.

User's new text: Miller

System action: Partition found containing the names

M.Muller

W.Muller

H.Muller

P.Mollard

S.A.Miller

No match is good enough (system must be careful with names), so the user is shown the list, headed by the best match, S.A.Miller.

User: Chooses S.A.Miller.

Note on response time: operating upon a disk file containing some 2,500 labels, the program's responses in the above exchanges are usually instantaneous (on a 360/67 time-sharing a fairly heavy university workload).

Formally, the result of the matching processes described is a set of points in the supergraph (see Chapter 4, section 1). We must now describe the file structures which support the much simpler function $f:N \rightarrow L$, i.e. mapping points onto labels; the lines, A, of the supergraph; and the searching for partitions, given their codes.

2. File organization

It may seem inelegant to talk in terms of files when we are considering the representation in storage of a labelled graph. However, we shall continue to use this terminology simply to serve as a reminder that whatever the design philosophy of a small-scale experimental system, the

designer is under some obligation to demonstrate the feasibility of his methods in an operational environment; and in reference retrieval that involves large quantities of data. We have, therefore, chosen to write a program which processes a graph structure stored on magnetic disk (i.e. in a file), rather than in main memory, even though the test data occupies a mere 360,000 bytes. In fact, the program runs under the supervision of the Michigan Terminal System (MTS) and all file processing is achieved using the standard data management services provided by that operating system (MTS,1973).

Corresponding to each point in the supergraph there is a node record on the disk. What we have, until now, referred to as a point is the address of a node record in the file. A set of points is an aggregate of addresses. The arrangement of such aggregates in storage varies, depending upon patterns of access to members; consecutive storage, linked lists and hash tables are all used. To return to the node record, it consists of two functional components:

(i) the label - see Chapter 4, section 1.1,

(ii) the set of points adjacent to it in the supergraph.

The second component carries the information specifying the set of lines in the supergraph. It is a redundant representation because each line is represented twice: once at each end. In the jargon of data structuring, the nodes are doubly linked. However, as is remarked in Chapter 6, section 2.1, the representation makes for efficient processing in this program. The parts of a node record are contiguous in storage, and there is a

large variation in the size of the records.

Access to the node records is, so far as the file management software is concerned, "random". In the course of the present experiment, the file has remained static since the final stage in its creation. Nevertheless, the design pays attention to the need, in "real life", for frequent updating. We must allow for addition and deletion of both points and lines, and also for amendment to labels which may bring about changes in the lengths of records.

For simplicity of programming, one would like to handle the supergraph in one level of randomly addressable memory. This can be achieved by building a very large, paged, virtual memory (one should think in terms of 100 million bytes for a useful field-oriented document collection). Virtual storage access methods exist for processing files on direct access devices such as disks (e.g. Murphy 1972, Organick 1972). Unfortunately, one cannot afford to forget that there is a paging mechanism, particularly when the virtual memory is so large. Firstly, one should take into account accessing patterns, and secondly, if data structures in the memory refer to each other, as ours do, one should take care in the design of record updating schemes which shift the position of the record in storage. Bobrow & Murphy(1967b) discuss these problems in connection with their implementation of BBN-LISP. In their case, it was important to implement the CONS (list constructor) function carefully. Since, in LISP, lists are nearly always accessed linearly, CONS should extend existing lists within the same page whenever possible. This policy influences garbage collection (the periodic amalgamation of free space

needed in dynamic storage allocation programs). List cells should not be moved from one page to another because that would ruin the effect of all the careful CONStructing. After collection, then, free storage is still distributed throughout the pages rather than being completely amalgamated. Many of the factors influencing the design of list processing systems for virtual memory are relevant to our file design, although it has been difficult during the program's evolution to anticipate the access sequences in graph processes, so that page swaps can be minimized. We should certainly need to tidy up the storage quite frequently if the variable-length records were being modified, and the reorganizations should preferably be truly local in their influence.

Node records are variable-length regions within large fixed-length blocks (4096 bytes). They have addresses which are invariant under storage reorganization within the block. The composition of a record address is as follows:

```
(block number:integer 0..2^16-1;
 record number:integer 0..255;
 record type:integer 0..255)
```

"Block number" identifies the block within the file (i.e. the page) containing the record. "Record number" is a number allocated serially when the record is added to the block. "Record type" is the type of the label in the addressed node record (see Chapter 4, section 1.1): there are many occasions when it is sufficient to know a record's type without needing its contents, and this small field can save a disk access. A record address is packed into

a single computer word (32 bits). If a record's position within the block is changed, its address remains the same and it is therefore not necessary to access all the other records with pointers to it.

Figure 14 shows the organization of a block and illustrates the addressing mechanism. Access within the block is through a two-level index, itself in the block.

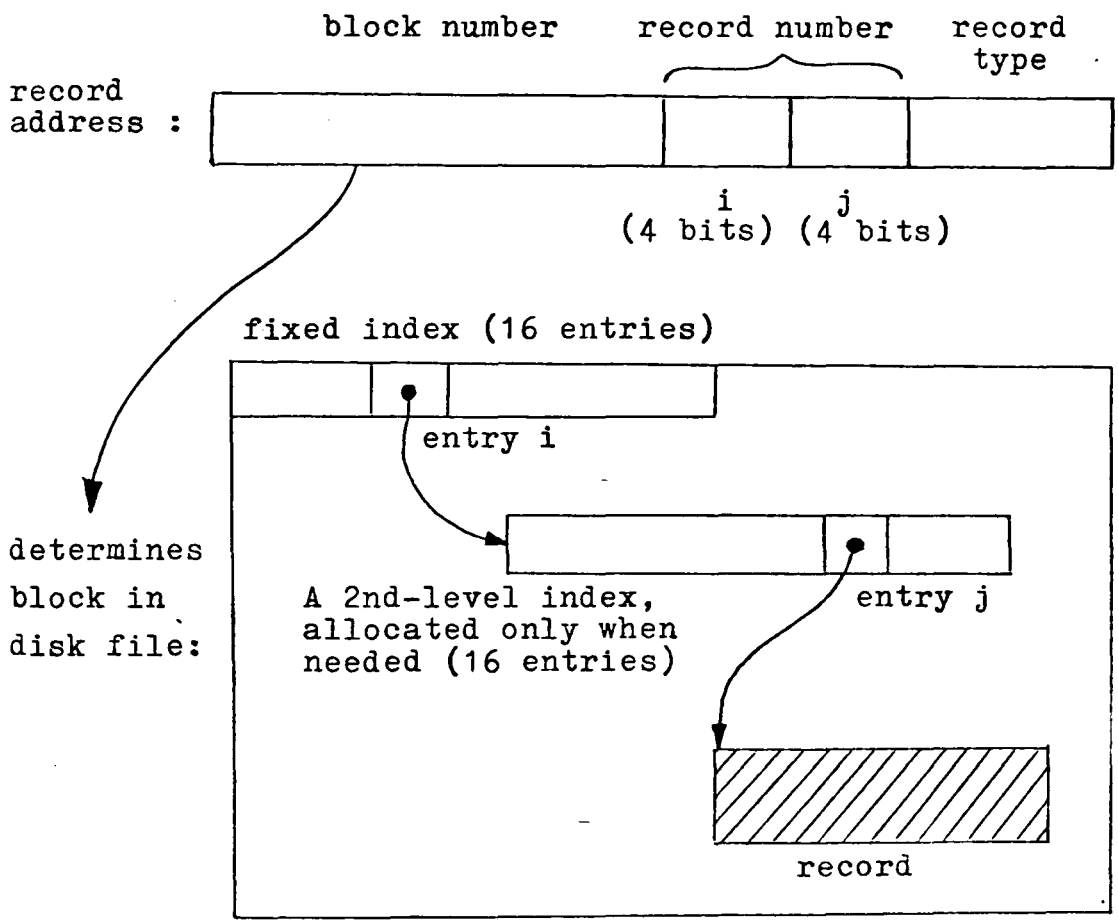


Figure 14. The record addressing technique.

At retrieval time, the work required to go through the index is insignificant in comparison to that involved in finding the block. Blocks are initialized with a fixed index full of null pointers, and no 2nd-level indexes.

When the first record is stored, a 2nd-level index is created whose first element points to the record. The first entry in the fixed index is set to point to the newly allocated index. As more records are added to the block, new entries are added to the 2nd-level index until it is full; then another 2nd-level index is created and a corresponding entry made in the fixed index; and so on. During execution of the program, the blocks of the file are paged into a set of buffers in main memory using the "least recently used" algorithm for displacing pages.

The format of the addresses gives an upper limit on the file capacity of 2^{16} blocks, each with 256 records, i.e. 16,777,216 records. For reasons given in section 2.2, below, not all of these are node records, but at least half can be, and that would be adequate for a large bibliographic data base. Of course the present block size imposes a limitation on the number of records per block, namely 120 of the smallest possible records; however, the block size could be increased.

It should be emphasized at this point that short cuts have been taken in implementing the system, particularly in the area of file handling, in order to speed the programming task. It is beleived, however, that the essential principles for a viable design for a very large file are present. One such short cut is that the record length is arbitrarily limited by what will fit into a block, since no provision has been made for overflow from one block to another.

The almost exclusive reason for the existence of very large records in this file organization is that a few

nodes will be the centres of large stars in the graph, involving up to 10^5 nodes. They correspond to the heavily posted terms in a co-ordinate indexing system (where they also cause problems). At the file organization level, one answer is to fragment the record and chain the segments together (Carville et al,1971 consider this type of technique). Increasing the block size mitigates the problem. At the higher, application level, the solution may be to prohibit such records, just as indexers might begin to use a set of more specific terms if it were found that one had become overused.

With the exception of the organization of the partitions of labels, where access patterns were easily predictable, little attention has been given to the problem of distribution of node records among the blocks with a view to minimizing the number of page faults (file accesses) during a search. Techniques exist to form clusters of references with the main aim of reducing the number of records which should be examined in a search (Jardine & van Rijsbergen 1971, Crouch 1973, Rettemeyer 1972). One might order document node records in the file such that members of the same cluster occupied neighbouring pages, or blocks of the file. To a large extent, it would be possible to put subject and name node records in the same region of the file, because the cluster definitions are all founded on similarity of descriptor sets associated with the documents. However, these clusters are collection-induced, and our point of view is that user-induced clusters are not the same, though they are clearly similar. The one access pattern which has emerged is that of obtaining the records

for points adjacent to one recently accessed. This suggests trying to minimize the sum:

$$\sum_{\{n,m\} \in A} |B_n - B_m|$$

where, as in Chapter 4, section 1, n and m are points in the supergraph and A is the set of all lines. B_x is the block number in the node address corresponding to the point x. A method for obtaining an arrangement of records which approximates the optimum might follow the general pattern suggested by Jardine & van Rijsbergen(1971) for clustering large collections. One first minimizes the sum over a small, carefully selected subset of the nodes, and then stores successive node records in blocks whose position is optimum so far as the new nodes are concerned. If one had to add the record for a point adjacent to some set, S, of points already filed, one might determine the block, B, in which to store the new record by minimizing

$$\sum_{i \in S} |B - B_i|$$

Any algorithm based on this will, of course, be complicated by having to cater for the possibility that the ideal block, B, is full.

It is usually assumed that when access to a file is "random", a large block size is wasteful of buffer size and quantity of data transferred. The speculations in the immediately preceding paragraphs seem to indicate that, on the contrary, for a system of this type, there is a great deal to be said for large blocks.

2.1 File processing within MTS

MTS (the Michigan Terminal System) is a time-sharing operating system designed to run on IBM 360 and 370 computers (MTS,1973). It enables the computer to be used simultaneously by many people operating a variety of keyboard terminals. Users may create files for their personal use, and editing facilities are provided. Most commonly, a user will have a few small files on public disk volumes containing programs under development, frequently used data, and so on. The disk units used on the Newcastle University machine are IBM 2314's (IBM, form A26-3599), which consist of a number of drives (up to 8) upon which disk volumes can be mounted, interchangably. A disk volume has a maximum capacity of about 29 million bytes. The read/write mechanism is of the movable-head variety. The tracks have a capacity of 7294 bytes, though the full capacity is rarely used, because some space is taken by inter-record gaps. There is a track overflow mechanism, so that records (i.e. physical, as opposed to logical records) need not be constrained to lie within a single track.

MTS supports two distinct file types - line files and sequential files (MTS,1973). A line file is an indexed sequential file in which the keys must be numerical. The lines, or records, can be of variable length, which may not exceed 255 bytes. Access, both for reading and writing, may be either sequential (i.e. in line number order) or random, by specifying the line number. With the facilities available for handling them in MTS, line files are extremely versatile and very convenient to use on-line for

tasks such as program development. This organization is not so suitable, however, for storing large blocks of data and implementing a paging algorithm for them. The sequential file organization provides a better basis. Sequential files stored on a direct access device, such as a disk, are processed using a set of pointers, which indicate the position of the next record to be read, where the next record should be written and where the end of the file is. Normally, one would go through the file consecutively and the pointers would be updated automatically. But a very limited form of direct access is possible: at any time, the values of the pointers can be saved, and then used to replace the current ones at a later stage. (One is warned not to calculate the pointer values, so that programs remain valid when modifications are made to MTS file software).

The test file is a standard MTS sequential file, on a disk volume, whose records are all 4096 bytes long (that is our block- or page-size). There are 80 blocks, numbered 0 - 79, and the relationship between block numbers and file pointers is set up in a table at the beginning of each run, by scanning the whole file sequentially noting the read pointer value before each read operation. This implementation has been perfectly adequate for experimental purposes, but in a full-scale system a specially designed (but relatively simple) data management package would be desirable.

2.2 Partition organization

It will be recalled that a partition is a set of points

whose composition is determined by the code obtained by compressing their labels. The compression of incoming labels induces a true partitioning of the points; in other words, the partitions are mutually disjoint and cover the whole set of points.

The representation of a partition in storage is simply an array of node record addresses contained in a partition record (see figure 13 in section 1.1 again; there is a correspondence between partition records and the square nodes). Partition records are stored in the file that we have just described, along with the node records; and are addressed in exactly the same way. As far as possible, it is arranged that a partition record and the node records to which it points are all in the same block (here is a case when we have known the accessing pattern all along). It now remains to describe the method of finding the partition record address, given the partition's type (name or phrase) and code.

A hash table is used and, since it is potentially very large, it is held in a disk file and searched from there. In fact, another MTS sequential file is used, also with 4096 byte records which are paged into main memory. There is a large literature on hashing, otherwise known as scatter storage or key-to-address transformation (Knuth 1973, pp506-549 and Morris 1968 are good accounts). The technique has been used in file organization for some time (Buchholz 1963, Lum 1971), and in bibliographic work it is not uncommon (Murray 1970, Higgins 1971, Bookstein 1972). We shall not include a general discussion of the topic here, but merely describe the way in which hashing has been used

for seeking partitions.

The search key consists of a type indicator (there are two possible types, which are represented by 0 and 1) and four characters from the set { space and letters A - Z}, which are represented by the numbers 0 - 26. The key is passed to the hashing function as a 21-bit binary number, K: 1 bit for the type and four 5-bit numbers for the code. The hashing function is as follows:

(i) The 21-bit key is squared to give S,
i.e. $S \leftarrow K^2$

(ii) Bits 0 - 19 are combined with bits 20 - 39 to give a 20-bit virtual hash address, V,
i.e. $$V \leftarrow \left\{ S \neq \left[\frac{S}{2^{20}} \right] \right\} \pmod{2^{20}}$$
($\neq$ denotes the exclusive or operation on the binary representations of its two operands)

(iii) The least significant n bits of V form the real hash address, R,
i.e. $R \leftarrow V \pmod{2^n}$, $n \leqslant 20$

R is used to address a table of 2^n entries. The value of n can change during the life of a growing file.

A non-empty entry in the hash table contains a virtual hash address and a partition record address, but not a copy of the key. The table is searched for V, starting at entry R and using a linear scan with increment 1 in cases of collision of real hash addresses. The reason for using this, the simplest overflow technique, even with its clustering problem, is to avoid as much as possible the costly crossing of page boundaries in the table. Collisions of virtual hash addresses are not detected, because the keys

are not recorded in the table (for reasons of storage economy). The result of the latter type of collision is that partitions are merged. There is no real loss of information here since the total contents of a partition are not indiscriminately selected by the program. Morris (1968) explains the concept of virtual hash coding: so long as n is significantly less than 20, the table is equivalent to a much larger, very lightly loaded table referenced directly by V, in which collisions should occur relatively rarely. The reason for using the virtual hashing idea was the ease with which a table can be doubled in size without the need to re-hash the whole file or change the hashing function (Bays,1973 goes into the problem of extending hash tables by re-hashing). When the real table is loaded beyond acceptable levels (say, more than $\frac{3}{4}$ full), the file containing the table is doubled in size and n is increased by one (thus absorbing a further bit from the stored virtual hash address into the real address). The existing entries are redistributed (keys are not needed for this), and values of R for new entries are found using the new value of 2^n.

The size chosen for the virtual hash table (2^{20} = 1,048,576 entries) limits the size of the real hash table. Ultimately, as the number of partitions grows, n will reach 20 and the virtual table will coincide with the real one. In this case there will be no collisions in the real table; partitions will simply be merged. The compression algorithms described earlier in this chapter can produce 54,000 name codes and 531,441 phrase codes; so the key space has 585,441 elements. We can get an approximate idea

of the extent of partition merging by assuming that the hashing function assigns the keys to table entries according to a Poisson distribution, which is reasonable if the function is a good "randomizer". The probability that any entry has been assigned k keys is

$$P(k;\lambda) = e^{-\lambda} \cdot \frac{\lambda^k}{k!},$$

where $\lambda = Kp$, the product of the total number of keys assigned, K, and the (uniform) probability, p, that any particular key will be assigned to any particular entry. If the table size is N, then $p = 1/N$, so $\lambda = K/N$. The probability that a typical entry is empty (k=0) is $P(0;K/N)$. Now, since K is large (585,441) we can invoke the Law of Large Numbers and say that the expected number of unoccupied entries is

$$N \times P(0;K/N) = Ne^{-K/N}$$

and that the expected number of occupied entries is, therefore,

$$N(1 - e^{-K/N}).$$

When $N = 1{,}048{,}576$ and $K = 585{,}441$, this formula works out to be close to 449,000. (This does <u>not</u> imply that the final table size - 2^{20} - is twice as large as it need be, because the <u>actual</u> number of occupied entries might lie between 524,288 (2^{19}) and 585,441). One can, thus, expect the partitioning scheme to work for files having order 10^6 node records, and that represents a very large field-oriented document collection.

We conclude with a few statistics concerning partitions formed in the test file. Further details and an account of

the test file will be found in Chapter 7, section 2.

(i) Number of node records:

Document type	225	
Name type	537	
subject type	1905	
	2667	total

(ii) Number of partitions:

containing 1 node	2373	
containing 2 nodes	113	
containing 3 nodes	14	
containing 4 nodes	4	
containing 5 nodes	2	
	2506	total

Note 1. Three partitions (all with two nodes) were formed "erroneously" by the phrase compression algorithm, in that dissimilar phrases generated the same code: *

{<Blood volume receptors and ... >
pulmonary diffusing capacity

{morphine
marihuana

{chimpanzees
cephalosporinase

Note 2. Four pairs of partitions (all with one node) were merged as a result of collisions in the virtual

---

* The same effect, of course, can cause occasional erroneous matching during a search. One medical user typed 'DEFORMATION' during a trial run, and was asked by the program if he meant 'MEDIAN RHOMBOID GLOSSITIS'. Before the present author could explain what had happened, the user exclaimed that he could see why the program had chosen that term: median rhomboid glossitis is a deformation of the tongue!

hash table. One would expect 3·01 merges among 2506 partitions.

(iii) Hash function performance:

Table size	4096	(2^{12})
No. of entries	2506	
Load factor, α	·612	(= 2506/4096)
No. of collisions (in real table)	734	

Distribution of search length, by linear probing, over all partitions:

no. of probes	no. of keys	no. of probes	no. of keys
1	1772	12	8
2	350	13	2
3	168	14	1
4	82	15	1
5	42	16	2
6	27	17	1
7	15	18	1
8	10	19	0
9	13	20	0
10	7	21	1
11	3	over 21	0

All partitions can be located once in a total of 4210 probes; average 1·68 probes per search. In 2506 searches, page boundaries in the table were crossed 8 times. Knuth(1973, p521) and Morris(1968) give equivalent formulae for the theoretical average search length for successful searches, using linear probe open hashing:

$$\frac{1}{2}\left(1 + \frac{1}{1 - \alpha}\right),$$

where α is the load factor, ·612 in our case. The value yielded by the formula is 1·79.

3. Summary

We have described the file organization and access methods which support the interactive reference retrieval program. The object of implementing an experimental system in this way has been to ensure that the type of retrieval dialogue proposed will not break down for want of techniques for handling involved structures in bulk storage, which are viable in the on-line situation. No attempt has been made to review file organization techniques - a topic which has received a great deal of attention in recent years. Dodd(1969) has written one of the better tutorial reviews, while Senko et al(1973) have recently contributed an outstanding three-part article on the subject. Lefkovitz(1969) has written a well known text which gives a broad view of file design for interactive programs.

The two important features of the file organization are these:

(i) It is possible to reach a pertinent point in the structure without being able to reproduce, exactly, the vocabulary of the system. This is done by lexical partitioning. It is not claimed that the algorithms are optimal; though they are based on the empirical results of others. Further experimentation could well lead to great improvements in performance, but one is in danger of meeting some of the fundamental problems of information retrieval, namely those involved in obtaining matches between mental concepts through the use of symbols (in our case index terms). We are concerned at the moment, with only one part of the system and tackling those problems here would

constitute recursion in the system as a whole.

(ii) The data base is regarded as a paged memory in which records (or data structures) within pages are the addressable units, as opposed to the word, byte, or any other rigid storage cell. In this way storage management for dynamic data can be efficient.

An important problem that we have not been able to tackle adequately is that of arranging records to suit the access patterns. Some suggestions have been made on an approach to a solution, and significant performance improvements can probably be made, particularly for present day high capacity magnetic storage media. Reference retrieval systems are built for communities of users, and one should therefore design a data base which can be accessed efficiently by several users simultaneously. The construction of a paged, virtual memory as outlined in (ii) above, but also capable of being shared, is a topic that merits further study.

Chapter 6

IMPLEMENTATION

1. Programming languages

In this work, we have concentrated on an engineering approach to reference retrieval, as opposed to a theoretical one. Ideas for the design of an interactive, mechanical aid to bibliographic searching have been incorporated in an actual program. Even in its illustrative, prototype form, the program is substantial. It has also undergone extensive modification in its brief evolution. The previous system designing experience of the present author, and of many others, shows plainly that it is only too easy to underestimate the size of a system implementation task. Some attention was therefore given, at the outset, to the methodology of program design, and we shall discuss this aspect of the problem in this chapter.

It is traditional among documentation programmers to bemoan the fact that there are no really suitable programming languages, or that the machinery was not designed with their purposes in mind. Two discussions of this topic are given by Salton(1966) and Dolby(1971). If, however, we regard programming not as the implementation of existing solutions, but as one means of discovering solutions, it is not at all surprising that no satisfactory special purpose language has emerged. We shall not dwell long on this question here.

The well-known programming languages are frequently classified according to the types of application for which they were designed. Fortran, Algol 60 and the early

autocodes were intended to be used to specify numerical algorithms: the emphasis was placed upon concise means of writing arithmetic expressions and iterative processes. For symbol manipulation, such as is needed in processing text, COMIT(Yngve,1963) and SNOBOL(Farber et al,1964) facilitate character string (sequence) handling, and IPL (Newell,1961) and LISP(McCarthy et al,1962) provide list and tree-structure devices. COBOL is the most commonly used of all languages and is designed for commercial and administrative data processing, where the entities of interest are records and files. These are merely a few of the languages available; many others have been developed for more specific application areas: languages used for writing problem solving programs have recently been surveyed by Bobrow & Raphael(1974), for instance.

For experimental information retrieval work, we can benefit from facilities and means of expression present in all three of the broad categories of language mentioned above. IBM's PL/I sets out to combine them all, and it is expensive to use. On the other hand, it is inadvisable to write various parts of a program in different languages because (i) there are practical difficulties in combining translated code and communicating data, and (ii) program maintenance and documentation are much too complicated. A third approach is to use a low-level language, which avoids the problem by favouring no particular application; but rather the machinery being used. Finally, new facilities can be grafted onto existing, more general purpose languages; for example, list processing onto Fortran - SLIP (Weizenbaum,1963) - string processing onto Algol

(Johnson,1974), graph manipulation onto PL/I (Santos & Furtado,1972). In view of the wide range of programming language features which are potentially valuable in this application area, the choice of one particular language, augmented or not, is felt to impose undesirable constraints on the solution of problems.

More promising than any of these conventional programming methods are those founded on the concept of abstract data structures, as discussed by Hoare(1972) and by Earley(1971). A program is written in terms of objects which correspond to, i.e. are abstractions of, entities in the problem. Implementation of such a programmed solution is achieved by finding another concrete form for the data structures, this time oriented towards the machine, instead of the problem. Broadly speaking, this is the technique used in the present project. Languages which embody this philosophy include Algol 68 (Woodward & Bond,1974) with its mode and operator declarations, and Simula 67 (Dahl & Hoare, 1972) which permits a very flexible procedural definition of new objects using the "class" construction. These languages possess a more powerful generality than PL/I, in that the programmer regards a data structure as an abstraction of some aspect of his problem which has its own appropriate operators, rather than as a record or aggregate of fields, which is an implementation-bound way of thinking.

The implementation language chosen for the program described in this thesis is a low-level language for the IBM System/360 computers, which has an Algol-like structure: it is called PL360 (Wirth,1968; University of Newcastle upon Tyne,1972). The major benefit obtained from being

able to write statements corresponding to machine instructions is that the range of design concepts which one can contemplate is very broad. In addition, PL360 is a convenient language to use interactively on a heavily loaded university machine because it has a fast one-pass compiler. The disadvantages of low-level programming which are frequently cited - obscurity in the program, and inefficient use of programmer time - are largely overcome by the methods which we describe in the next few paragraphs. McCracken & Garbassi(1970) write:

> "With COBOL, or any similar high-level language, ... changes are relatively simple to make ... With machine-language programs there are actual examples of cases in which adding one more digit to a deduction has required weeks of reprogramming." - p81.

This comment is either a gross exaggeration or an unwitting observation of bad programming practice. (It is ironic that this claim should occur in an introductory text on COBOL, a language which certainly does not encourage good programming habits, even if it does have useful facilities for commercial data processing).

2. The structure of the program

The explicit aim of the method of programming given here is to make the bridge between problem and machine as clear as possible. The method also makes that bridge shorter. We benefit in three main ways:

(i) Program development is fast, at both the writing and the testing stages, because the risk of errors is low (by normal programming standards). It is important to have a low error-rate when implementing heuristics, where unsatisfactory output can be

attributed either to coding errors or to poor heuristics, thus adding to the complexity of testing. No precise measurements of programmer performance were made in this project, and there is no clear distinction between design and coding phases, but rough estimates can be made. The programs are written, first, in what we might refer to as a design notation, which is the basis for the PL360 coding. If we consider all tasks performed from (and including) the writing of the design notation to the acceptance of the PL360 coding as "correct", this particular programming job was done at a rate of about 100 PL360 statements per 8-hour working day. Only one inconsequential, and easily corrected, error was discovered in the complete final version of the program during some 300 dialogues with Thomas.

(ii) When our ideas on the problem change it is possible to identify, quickly, those parts of the program which will be affected.

(iii) Documentation of the program is aided by the method. The design notation provides a precise and well organized description of what the PL360 procedures do, and there is a close notational correspondence between the two. In fact, the description of the program Thomas given in Chapter 4 follows the design notation, and its writing was aided considerably by having that specification to hand.

The structure of the program is, on the whole, hierarchical; we have used the "top-down" approach advocated by Dijkstra(1972). There are many interesting discussions

on the topic in the literature (Wirth,1971; Henderson & Snowdon,1972; for instance), and Snowdon(1974) has put forward an interactive program development tool which encourages the conscious use of the principles involved in clearly structured programming. We shall not therefore embark on a lengthy discussion here, but show how the principles have been applied in building Thomas.

2.1 The "top-down" approach in use

We should like to write programs that have a structure which makes them readily understandable. The most desirable attributes that an algorithm (i.e. a procedural program) should have to achieve this aim are as follows:

(i) It should be seen to be of the form
"First do A, then do B, then do C, ... ",

(ii) It should be short,

(iii) The data objects that it handles should resemble, in the notation, the "problem" entities of which they are abstractions.

A programmer can be confident that such an algorithm does what he intends it to do. The first attribute can be achieved, very nearly, with the Algol program control devices: procedure calls, for, while and repeat statements (for making loops into "do X"), and if and case statements (for alternative paths)*. To achieve the second attribute, the programmer should build his system out of short (e.g. less than a written page) procedures - modular programming. The third attribute of a clearly written procedure can be

---

* A good source of information on Algol 60 is Dijkstra (1962). It includes a copy of the "Report on the algorithmic language Algol 60". For later suggestions, including the case statement, see Wirth & Hoare(1966).

achieved by inventing data types as needed, together with appropriate operators and programming constructs. It leads to obscurity if we use an integer type of variable to denote, for example, a file address consisting of three numbers which we have decided to store, packed, in one computer word.

These considerations, added to the fact that the implementation language, PL360, is Algol-like in its structure, led to the choice of Algol 60 as the basis of the design notation. Because this notation was intended to be open-ended, there has never been any intention to automate the generation of programs from it. So, the technique amounts to writing programs in an extending Algol, and translating them by hand into PL360.

Let us follow part of the development of program Thomas. Some of these procedures are described in Chapter 4, starting in section 3, and the reader may wish, occasionally, to refer back to the accounts given there. We shall start with the requirement to write a program that creates and maintains a model of its user's interest, to help him search for references on a particular topic. To start with, we simply state the requirement a little more formally, as a process:

```
procedure TOPIC_SEARCH;
begin SET_UP_MODEL;
      repeat IMPROVE_MODEL
      until USER_SATISFIED
end.
```

Firstly, an initial model will be created by a process named SET_UP_MODEL. The model will be modified in stages, as the dialogue proceeds, until the user has seen as much as he wants. IMPROVE_MODEL is a process which results in a change in the model, and USER_SATISFIED is a Boolean-valued function which determines whether the user has commanded the dialogue to stop. All the symbols in upper-case letters are names of separate processes. The procedure TOPIC_SEARCH acts as a manager which has delegated the various jobs and which coordinates the activities of its subordinates. Most of the processes introduced would, at some stage, be defined in the Algol design notation in terms of further processes. Each Algol procedure is finally translated into PL360 using conventions which evolved early in the project. Some processes named in the Algol procedures are close enough to the capabilities of the machine not to need an Algol definition themselves. In these cases, we can either write a procedure directly in PL360, or put the appropriate code in-line when translating the calling procedure. There follows an outline of the PL360 version of TOPIC_SEARCH. Because the reader is assumed not to be familiar with PL360 nor with the conventions referred to above, this will be the only example given and, even so, it will be simplified. Procedure names are abbreviated to eight letters in the PL360 translations:

```
global procedure TOPICSEA(R14);
begin
      external procedure SETUPMOD(R14); null;
      external procedure IMPROVEM(R14); null;
```

```
external procedure USERSATI(R14); null;

    ┌──────────────────────────────────────────┐
    │ 5 lines of declarations and coding for   │
    │ storage management, including a          │
    │ declaration for the local flag: stop     │
    └──────────────────────────────────────────┘

    SETUPMOD;
    RESET(stop);  R2:=@stop;
    while ¬stop do
    begin  IMPROVEM;  USERSATI;  end;

    ┌──────────────────────────────────────────┐
    │ an instruction concerned with storage    │
    │ management                               │
    └──────────────────────────────────────────┘

end.
```

USERSATI is a translation of the Boolean function USER_SATISFIED and, by convention, will put its resulting value in the flag addressed by register R2, namely stop. The PL360 while statement is then equivalent to the Algol repeat.

The elaboration of initialization processes like SET_UP_MODEL should normally be deferred until more is known about the central processes. We therefore move on to IMPROVE_MODEL. In the mechanism we are designing, the model is to be adjusted according to the user's input, which will normally be his response to the program's last display. The repeat statement in TOPIC_SEARCH takes care of the iterative aspect of the dialogue. The tasks of giving and receiving messages, and of changing the model are delegated to IMPROVE_MODEL. Let us first define the process quite vaguely:

```
begin
      read a message from the user;
      use it to influence the state of the model;
      make a response to it
end
```

The intention is to invoke three more procedures to perform the constituent processes in this definition. Those procedures will be regarded as the definitions of the meanings of the phrases, and they will be written independently. There is, however, a link between them which must be represented in the more formal definition - namely the message, occurring as the pronoun "it" in the second and third phrases. We must introduce an abstraction of a message: a data type, one instance of which will be made available to the procedures to formalize the link. The message that the user types will be a simple sequence of characters, but we judge that it will probably be best to structure it in some way on receipt. We therefore invent a name for the data type, message, and postpone defining its properties until we know more about the way we wish to use it. We can write the Algol definition of IMPROVE_MODEL now.

```
procedure IMPROVE_MODEL;
begin message m;
      m:= GET_USER_MESSAGE;
      INFLUENCE_STATE_OF_MODEL(m);
      RESPOND_TO_USER(m)
end.
```

A note about the implementation of the model: because it forms the basis of the system, the model is regarded as global to every procedure. Naturally we know quite alot about the structure of the model, but we do not yet need to make it explicit in the programming.

GET_USER_MESSAGE, which is a function of type message, must process the input in a way that is not yet decided, so we defer its definition. Here is an informal definition of INFLUENCE_STATE_OF_MODEL:

```
begin
    update the performance figure in the model, according
        to the user's reaction;
    prune rejected points from the context graph in the
        model;
    add selected points to the context_graph;
    find and add explicitly requested points;
    make sure the context_graph is connected
end
```

Each process, except the last, uses some information which it is assumed can be derived from the user's message. We invent a set of functions which require a message as argument and yield just the types of values most suitable for feeding to the procedures that we shall invoke to perform the required processes. They are called selector functions. The selector functions called by the procedure which follows are called reaction, reject_list, select_list and request_list. We still do not have to decide precisely in what structure the data they return should be. For each data type that we invent, we keep a record of its selectors

and make a note of their types when they are known.

```
procedure INFLUENCE_STATE_OF_MODEL(m);
message m;
begin
      COMPUTE_SCORE(reaction(m));
      PRUNE_CONTEXT(reject_list(m));
      ADD_TO_CONTEXT(select_list(m));
      FIND_NODES(request_list(m));
      UNIFY_CONTEXT_GRAPH
end.
```

The programming continues in this way, and we shall show a little more below. However, we pause at this point to remark on an omission in the definition of INFLUENCE_STATE_OF_MODEL, which did not, in fact, come to light until the Boolean function USER_SATISFIED (see the definition of TOPIC_SEARCH, above) was elaborated. The problem then was to decide how the program should determine that the user had seen enough. The dialogue would have been clumsy if, before accepting a substantive message from the user, the program had to ask him if he wished to stop. Much better that he could say "stop" in place of the normal message. We required, therefore, a means of recognizing the stop message by GET_USER_MESSAGE, and of passing the request on so that INFLUENCE_STATE_OF_MODEL and RESPOND_TO_USER should not execute in the normal way, and so that USER_SATISFIED should return the result true. The method chosen was to record in message a special value for reaction, denoted by STOP, and modify INFLUENCE_STATE_OF_

MODEL to the form given in section 3.2 of Chapter 4.

We return to work down the hierarchy of processes a little further, to illustrate the handling of the model and of graphs at this high level of description. A mathematical description of what we mean by the supergraph and the model are given in Chapter 4 sections 1, 1.1, 1.2 and 2: it is, very largely, in terms of sets of points. We define FIND_NODES:

```
procedure FIND_NODES(requests);
query list requests;
begin global point set explicit_requests, inhibit_list,
                        context_graph;
      point set addresses;
      addresses:= LOCATE_NODES(requests);
      explicit_requests:= explicit_requests ∪ addresses;
      inhibit_list:= inhibit_list - addresses;
      context_graph:= context_graph ∪ addresses
                              ∪ STARS(addresses)
end.
```

There are several remarks to make about this procedure:

(i) The parameter, requests, is the value returned by the selector function request_list acting on a message. We have given this type of data a name, query list, but have still not needed to decide on the details of its structure, except what is implied by the use of the word list; i.e. that the query's are organized in a sequence, so that the order in which the user provided the texts is maintained.

(ii) We have invented another data type, point, and have specified that the aggregates of point's mentioned should display the properties of sets.

(iii) For manipulation of sets, the operators ∪ (union) and - (asymmetric set difference) have been introduced. In the PL360 translation, one would expect these to be implemented by procedures, but we shall only become concerned with that when a concrete representation for sets is chosen.

(iv) Because the Algol procedures are defined independently, it is necessary to state that the model components explicit_requests, inhibit_list and context_graph are the same variables as those accessed by other procedures. The symbol global is used for this purpose.

(v) The procedures LOCATE_NODES and STARS are point set valued functions.

We now define STARS, a procedure which computes a set of points adjacent in the supergraph to the points in its argument set.

```
point set procedure STARS(centres);
point set centres;
begin global point set inhibit_list, context_graph,
                       check_tags;
    point set result;
    point p,q;
    result:= empty;
    for each p in centres - check_tags do
    for each q in LINKED_TO(p) do
```

```
        if q ∉ inhibit_list ∪ context_graph ∪ centres then
                        result:= result ∪ {q}
    STARS:= result
end.
```

The construct for each ... permits us to specify that a process should be performed for each member of a set, without straying into implementation questions concerned with the order of the elements in storage. The operator $\notin$ means "is not a member", and the brackets $\{\ \}$ turn their contents into a set.

To access neighbouring points in the supergraph we use the procedure LINKED_TO:

```
point set procedure LINKED_TO(p);
point p;
LINKED_TO:= node_links(NODE_AT(p)).
```

The procedure NODE_AT is responsible for finding the data object containing information relating to a point in the supergraph(the label - see Chapter 4, section 1.1 - and the set of adjacent points). Objects of this type are referred to in other parts of the program as node's. Here, we need the set of points adjacent to the node, and decide that they should be available through a selector function called node_links. We have not, at this stage, explicitly considered implementation of the supergraph (although, of course, that question is certainly in the back of one's mind). It may be noted, however, that we already have a hint of some of the details of file organization given in

Chapter 5, section 2. The point's are beginning to look like file addresses, and to make the selector node_links (and thus the procedure LINKED_TO) work fast, the data structure retrieved from the file by NODE_AT should contain the addresses of all adjacent nodes.

We shall leave the program development at this point. Regarding the treatment of sets in the "Algol" definitions, it should be pointed out that more elegant notations can be used if non-procedural programming is adopted (e.g. Elcock et al,1971).

Top-down programming is not an infallible method for effortless problem-solving. It is often necessary to know how the machine will do a task before writing the high-level procedures, in order to obtain a satisfactory breakdown of the design. Bottom-up programming starts near the computer and works up towards a solution to the problem. It is often necessary, even when the approach claimed is top-down, and is sometimes explicit but more often implicit. Subconscious bottom-up programming probably permeates every stage in a feat of top-down programming; it is the programmer's use of his experience. Conscious, though not necessarily explicit, bottom-up programming occurs when we decide, for example, whether a search is best done by hashing or binary search, or when we choose one Algol definition rather than another because it can be rendered easily into PL360. There is, however, a basic difference between the two approaches. In constructing a system, we do not know the solutions to all our problems in advance, and it is natural to start by working from the top. The procedures we write will then be those actually needed in the

system.

2.2 Data structures

Once again, we must avoid a general discussion and refer to Hoare(1972) for a comprehensive treatment of the subject of data structuring. What we require for the top-down programming method we are using is the ability to invent any data structure, and not necessarily all at once. In the procedure, IMPROVE_MODEL, in the previous section, it is acknowledged that we need a structure of a type called message, but no further details are given - rightly so, because they would only obscure the meaning of the process. In INFLUENCE_STATE_OF_MODEL, certain aspects of the message type are introduced: reaction, reject_list, select_list and request_list. The attributes of each of these come to light at various stages in the development, as does the need for yet more components of the data structure representing a processed user input string.

In general, if we wish to introduce a concept as a structured aggregate of information, then we just invent a name for it (made into a basic symbol by underlining it) and use it as a data type in a declaration of one or more instances of that concept. When we wish to get at some of the information which we understand to be part of the concept, we invent a selector function, which selects data of a particular type and in a particular semantic role from the total abstract object. As programming proceeds, details of the original concept are filled in, and thus a collection of selector functions is built up. At any point the entity is understood in terms of the

collection of selectors invented for it. When a collection of Algol procedures is translated, the new data structures are implemented simply by arranging for all the selectors to work easily - this usually means no more than setting out corresponding fields (or pointers) one after the other in a storage map.

Examples:

(i) node N; Many instances of this data type reside in the data base, and collectively define the supergraph.

selector functions:

node_kind(N) takes one of the values NAME, DOC, SUBJECT

If node_kind(N) = NAME,

node_name(N) is a string, representing a surname, and

node_initials(N) is a string, the initials of the forenames.

If node_kind(N) = DOC,

node_phrase(N) is a string, the title of a document,

node_ref(N) is a string, the location in a journal.

If node_kind(N) = SUBJECT,

node_phrase(N) is a string representing a subject term.

node_links(N) is a point set containing all the adjacent points (addresses).

In Hoare's terminology, this example is a discriminated union of Cartesian products - we have joined

into one data structure three composite structures ("products" of more elementary types). The selector function node_kind simply serves to discriminate between them.

(ii) string tree T; This structure is used to identify suffixes on words.

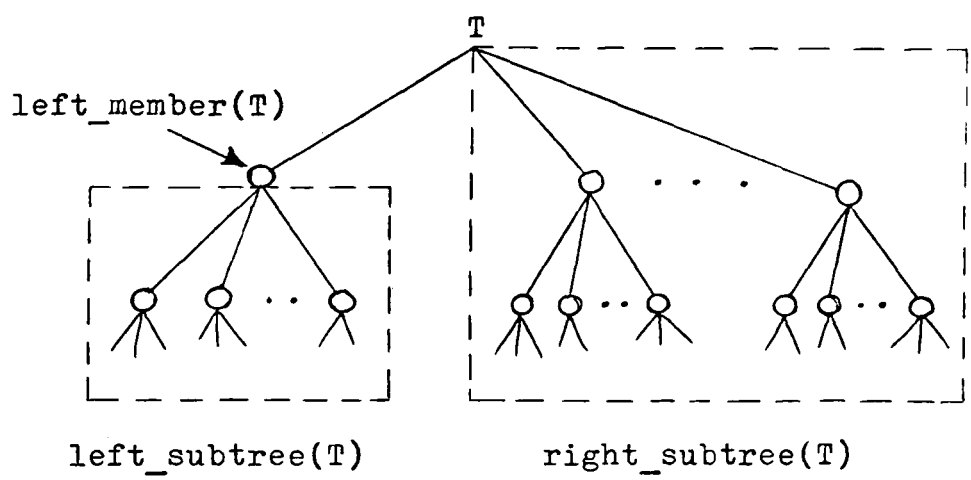
selector functions:

left_member(T) is a string,

left_subtree(T) is a string tree,

right_subtree(T) is a string tree.

A pictorial representation of a string tree is as follows:



It should be emphasized that this definition of a tree arose purely from the introduction of the selector functions in the program; it was not decided upon in advance.

2.3 Implementation of data structures

Hoare(1972) discusses in detail the considerations which influence the implementation of abstract data

structures. In a large system of procedures, we also have to decide whether to establish system standards for the various concrete data structures, or whether representations can vary according to local needs. Because data is passed from procedure to procedure in parameters and function values, standardization is the predominant policy in this project; there are exceptions. When storing structures with pointers (containing variable length strings, for instance) on the disk, the representation must be relocatable, so the pointers are stored as offsets from the start of the region (or "record"). At other times the links are absolute addresses for simpler access and, more important, so that we can incorporate certain existing structures into others simply by making reference to them, wherever they happen to be, instead of moving them in storage.

On the coding of data structures, we shall not go into the details, which are generally very straightforward, but merely remark that most structures have two components - a fixed part containing fixed length data and pointers, and a variable part containing such things as linked list structures and sequences of characters.

The representation of sets deserves mention, however. Sets differ from other abstract data objects in that there are no selector functions; all processes are specified, in "Algol", by means of set operators ($\cup$, $\cap$, $-$, $\in$, and so on) and the for each ... construct. Underlying these operations are four basic ones: (i) determining whether an element is a member of a set, which is a search operation, (ii) scanning a set, i.e. considering every member, (iii) adding an element to a set, and (iv) taking an element

away from a set. For sets that are frequently searched, a hash table representation is efficient. In this program, we have a limited number of global sets (in the model), which must remain accessible throughout execution of the program. If a point belongs to any of these, there will be an entry for it in a globally accessible hash table, indicating by means of a short bit vector which sets it belongs to. Elements can be added and removed very easily, but the scanning process is very inefficient. If the program requires to scan a set, a linked list structure is used to represent the set, sometimes in addition to the hash table representation in the case of global sets. These structures are kept in a large globally accessible storage area, with the exception of sets declared locally within the Algol procedures. The "node_links" portions of the node structures in the data base are put into the same area when called for. While they remain in that area they can be accessed through the same hash table that holds information for searching the global sets.

2.4 Use of storage

We have chosen an Algol program structure, so storage must be organized in a stack (except for that used for global variables). The stack must accommodate lists and any other volatile linked or variable length structures we care to invent. According to the conventions developed for this project, the stack is maintained in contiguous storage locations in virtual memory. A PL360 procedure is told where it may start to store local data, and before returning control must destroy its local variables by

adjusting the top of the stack downwards again. The details of the technique are different from, but compatible with normal IBM 360 subroutine linkage conventions to the extent that MTS library routines can be called without trouble.

The problem arises when the result produced by a procedure (corresponding to an Algol function procedure) or a new value assigned to a parameter is of unpredictable size. Conventions, making use of a second stack, allow the main stack to be handled in such a way that, while the fixed part of a resulting data structure is provided by the calling procedure, the called procedure is responsible for ensuring that, on return, the variable part of the structure is stored within the stack as known by the calling procedure.

3. Management and documentation of the programming

With 228 procedure and selector function names in the Algol definition of the program, it is inevitable that an appreciable amount of time had to be spent on managing and documenting them. The system has been constructed in several sections, typically defined by 15 - 20 Algol procedures. These are translated into a set of PL360 global procedures, and tested. Usually, there are calls on procedures which have not yet been defined and simple temporary substitutes must be written for these. Also a main program must be written to run the test.

A difficulty which arises when testing pieces of a file processing system is that large, complex, test data structures are sometimes needed. Construction of these by hand can be so laborious and error-prone as to be impract-

icable. To construct the data automatically often requires the definition of another part of the system, which in turn requires extra programs to independently check the data and validate the structural representation. To make matters worse, it is often not possible to define the data structure to be produced by the building sub-system before the processing sub-system has been written and its requirements are fully known. We must resort to a complicated ad hoc testing of the two sub-systems in parallel, in which quite alot of extra programming is necessary.

Program testing has been done on-line, and debugging has assumed a much less prominent place in the development of this system than is traditional in programming. Most errors cause PL360 compiler diagnostics and are simple slips in translation or typing. One subtle logical error in the Algol definition was due to the awkward ordering relation among English suffixes while they are still attached to the words. The first method of identifying a word's maximal suffix which was tried comprised reversing the letters in the word and searching a sorted reverse--suffix dictionary using the binary search technique. It cannot be done that way because the length of the suffix, if any, is not known until it has been identified. A tree searching method was used instead. When testing is "complete", the object modules are added to a program library, and the final version of the PL360 source in printed form and on punched cards is filed away.

An analysis of the means of implementation of all the procedures and functions called in the Algol-defined part of the program follows:

(i) Selector functions for a variety of data structures are implemented:

a) by simple reference to a field in a storage map	44
b) by minor manipulation	9
(sub-total)	53

(ii) Other functions/procedures are implemented:

a) by translation of Algol procedure into corresponding PL360 procedure	125
b) by small in-line code sequence (i.e. 1 - 5 instructions)	27
c) by definition directly in PL360	23
(sub-total)	175
(total)	228

In addition there are 53 PL360 procedures which have no Algol equivalent. These perform tasks such as storage management and set operators, and, like the Algol definitions, they are short and hierarchically organized.

The program documentation consists of the Algol procedures themselves, lists of all invented data types and their selectors, descriptions of the representations of data in the machine, and an index to procedures, recording how they are defined, which other procedures they call, which other procedures call them and, in the case of selectors, which data type they operate upon. This information has been found adequate for development. For example, if it is required to change the implementation of a data structure, one first makes a list of all its

selector functions. Looking them up in the index will yield a list of all the procedures which call them, and these will determine which PL360 procedures need be changed.

Questions of managing design and implementation decisions in a flexible way are considered by Parnas(1972). His answer is the concept of "information hiding", and his conclusion is

> "that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing." - p1058.

Clearly, Parnas' systems will be well-structured, but not hierarchically, from the top, down. His methods seem, at first sight, to be rather different from those described here. However, inherent in the system presently under consideration, there is a sort of dynamic modularization, which can have Parnas' desirable information hiding property when needed. We handle design decisions and document the system in such a way that modules (in Parnas' sense) can be temporarily assembled out of procedures, for specific purposes. Any retrieval criterion can be applied to the program documentation, in principle.

4. Summary

In this chapter we have given an account of the methodology of the implementation of the illustrative reference retrieval program, Thomas. The method described is not proposed as the only sensible one, even for experi-

ments, because a great deal depends on the programmer's past experience and what might be referred to as his taste in programming styles. Nevertheless the method has very useful properties for our purposes. The design and programming aspects of the job are not clearly separated, programming is quite fast and debugging is very fast, documentation is facilitated, and, as a result of all these, changes of mind on the designer's part are relatively painless. The technique is one interpretation of the top--down programming method (Dijkstra,1972); and we have, in this chapter, illustrated its use by quoting from, and commenting upon part of the actual development of Thomas.

Chapter 7

PERFORMANCE OF THE PROGRAM

1. General remarks

A great deal of the literature on reference retrieval is concerned with methods of evaluating systems: the basic measurable units and the performance statistics derivable from them. Firstly, we should distinguish retrieval performance and notions like efficiency and cost. We are concerned in this chapter with the former. Most workers in this field associate retrieval performance with a system's ability to pick documents which are relevant to the queries put to it. Consequently, most performance measures are based on the 2 × 2 contingency table showing how the system's relevance decisions compare with the user's. If, in a collection of N references, there are C relevant to a particular query, and the system retrieves L references, of which R are among the relevant ones, the system's performance in response to that query can be shown as follows:

	Retrieved	Not Retrieved	Totals
Relevant	R	C - R	C
Not relevant	L - R	N - L - C + R	N - C
Totals	L	N - L	N

Note that in any realistic collection, the value of C is not known; it is, in fact, the size of the set A that was introduced into the discussion in Chapter 2, section 1.1.

"Laboratory experiments" in reference retrieval (prominent current examples are the work of Sparck Jones and of Salton; important earlier work was done by Cleverdon) make use of a small document collection, a set of queries, preformulated or formulated by the system from natural language questions, and, for each query, the set of "relevant" documents. In other words, in these experiments, C is known, and the table can be compiled, completely, for each query. By adjusting some parameter of the system under test, the values of R and L are varied and, in order to assess the relative merits of different values of the parameter, the contingency tables obtained are summarized. A normalization technique must be combined with the averaging process so that systems, or variations in search strategy, can be compared. We can combine the figures obtained for a set of searches (by addition) and then normalize, presenting ratios (named "micro evaluation" by Rocchio,1971). Alternatively, and this reflects the viewpoint of the individual user, we can work out some ratios from each table first, and then average them ("macro evaluation"). The most commonly used ratios are called recall and precision. In terms of a single contingency table, these are defined:

$$\text{recall} = \frac{R}{C}, \qquad \text{precision} = \frac{R}{L}.$$

Combining the ratios over several contingency tables can be done in two ways:

$$\text{micro recall} = \frac{\sum_i R_i}{\sum_i C_i}$$

$$\text{micro precision} = \frac{\sum_i R_i}{\sum_i L_i},$$

or:

$$\text{macro recall} = \sum_i \frac{R_i}{C_i}$$

$$\text{macro precision} = \sum_i \frac{R_i}{L_i}.$$

In spite of the fact that recall and precision have been vigorously attacked as an unsuitable pair of measures (Fairthorne 1964, Robertson 1969, for example) they continue to be the most widespread criteria for retrieval system worth, probably because they correspond to the supposed aim of reference retrieval - to find as many as possible of the relevant documents, and to avoid picking up irrelevant ones in the process. They are even the most frequently used basis for evaluating fully operational systems, where C is unknown, and thus true recall is unobtainable. In these cases, methods have been devised for estimating recall, or using a similar ratio which, in comparative evaluations, provides an indication of recall (Lancaster,1969; McCarn & Stein,1967). The criticisms have been founded on the mathematical interdependency of the ratios and the validity of the averaging processes (which inevitably lose information).

Another type of criticism, for example that by Cooper(1973), is that measures depending upon (perhaps dubious) collection dichotomies are not necessarily related to system utility. The alternative is some form of subjective evaluation - the user attaches a value to the service he has received. In Cooper's proposal, the user states what price he would pay for a relevant reference, and how much he would pay to avoid seeing a non-relevant one. In a recent paper on this topic, Cleverdon(1974) has argued the need for the evaluation of the utility of information systems, while at the same time recognizing the power of recall, precision and other such measures in the laboratory. Cooper is criticized for confusing value with performance: it is conceivable that a system which performs very well, may be unusable, and therefore of little value, because people cannot easily express their information need in the required form. The distinction which Cleverdon draws is problematic, now that the enquirer can conduct his own search on-line. The user now plays a major role within the system itself. Is it still sensible to attempt an evaluation of the mechanical part of the system in isolation?

The diversity of views concerning methods of evaluating a reference retrieval technique is probably attributable to differences of opinion on the nature of the retrieval problem, and what qualities enquirers look for in a system. It seems sensible to seek measures which will enable us to state how well our program performs the tasks which we designed it to tackle. Rather than prolong the discussion of evaluation in general, therefore, we shall turn to the

attributes that should be tested in the retrieval method proposed in this thesis.

No matter how a library user approaches the literature, whether straight to the books, or through the most advanced retrieval system, he views a small part of the totality of literature. There is no doubt that some users, on some occasions, would like their view to be accurately and efficiently restricted to one small area. This requirement falls at one end of the search/browse spectrum, and our program, Thomas, may not be valued very highly by such users. Most searches, however, have an element of browsing in them (Herner,1970), and the user's view should include a certain amount that is peripheral to the strictly relevant. Imprecision in retrieval is not without value. It may increase the user's awareness of potentially interesting work or information sources, and can help him state or decide what really is pertinent to his own work. On the other hand, high recall is also not necessarily required by users. Cleverdon(1974) suggests that, "for many subjects, a recall ratio of 25% or less of the relevant documents will give a complete 100% recall of information." - p174. These points should be borne in mind when using recall and precision to describe the performance of an interactive retrieval system.

A major deficiency of the evaluation in this chapter is that it has not been possible to conduct extensive trials with real users. The scale of such experiments is beyond the resources of this project.

2. The test collection

The collection of references used to test the retrieval methods described in this thesis is a subset of the references added to the Medusa current awareness file (see Chapter 2, section 2.4.1) in September 1973. Out of about 19,000 references we have chosen 225. Firstly, searches conducted by medical scientists and biochemists, on the Medusa system, were selected if they had resulted in any retrievals from the September section of the file. By search, we mean the complete query formulation process, which may contain several "SEARCH" commands. All references so retrieved by Medusa, whether relevant or not, were selected. The important point about this method of selection is that we have queries and corresponding relevance judgements made by practitioners with genuine information needs. In addition, if the Boolean search strategies formulated with Medusa's aid were to be put to the subset, the output would be precisely the same. The figures so far:

1. Number of searches,

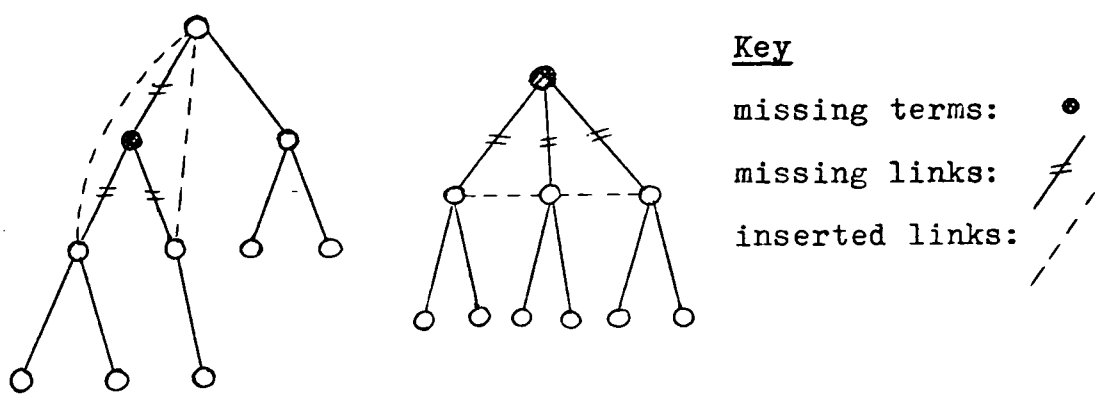
(i) retrieving no relevant references:	14
(ii) retrieving 1 or more relevant ref.:	32
Total	46

2. Number of different references: 225
3. Total no. of relevant references: 91

A network of records (the "supergraph") was built up from these 225 references. All the authors and index terms associated with the references were extracted from the Medusa files and linked to the document records. In the MEDLARS records used by Medusa, some terms are accompanied

by qualifiers (see displays in Chapter 2, section 2.4.1); the qualifiers have been dropped for Thomas' data base. The distinction between "print" and "non-print" terms is also ignored.

Using the MeSH (Medical Subject Headings) category structure, many links were inserted between index terms. There are several categories, each is a hierarchy of terms, and many terms occur in more than one place in this arrangement. Because there are a large number of terms missing from our test file, the tree structures were disconnected in places. Simple conventions were adopted for linking them up. The following picture illustrates the rules:



Terms were not linked if the shortest path between them had more than two lines.

Several thousand synonyms have been added to Medusa's dictionary. Those attached to terms already selected for the supergraph were included, except where the phrase compression algorithm (Chapter 5, section 1.1.2) would have caused the synonym to be recognized correctly. Many of the synonyms in Medusa are word permutations of the correct term, and our program can deal with these without the need to store them separately.

Here are some further figures describing the test file:

Number of references: 225

Number of authors: 537

Number of MeSH terms: 1357

Number of synonyms: 551

The distribution of the MeSH term postings follows (a posting is equivalent to a line joining a subject node to a document node in the supergraph):

Terms	No. of postings
HUMAN	150
MALE	87
FEMALE	84
ANIMAL EXPERIMENTS	77
ADULT	51
MIDDLE AGE	44
METHODS	40
TIME FACTORS	36
ENGLISH ABSTRACT	30
CHILD	30
RATS	27
AGED	27
ADOLESCENCE	27
CHILD, PRESCHOOL	20
MICROSCOPY, FLUORESCENCE	16
No. of terms	
4	15
3	14
6	13
5	12
3	11
6	10
8	9
15	8
7	7
19	6
35	5
46	4
83	3
238	2
864	1

Average no. of postings per reference: 15

Average no. of postings per term: 2·48

With the exception of MICROSCOPY, FLUORESCENCE, the named

terms in the above table are all check tags. (As we have already said in section 3.1.1 of Chapter 3, check tags are terms which the indexer must consider posting to each document). The remaining check tags in the test file are

AGE FACTORS	INFANT
CASE REPORT	INFANT, NEWBORN
CATS	IN VITRO
CATTLE	MICE
COMPARATIVE STUDY	PROGNOSIS
DOGS	RABBITS
GUINEA PIGS	REVIEW

There are a few more in MeSH which happen not to occur in the test file.

We should now satisfy ourselves that there is a substantial amount of overlap between the subject areas represented by the queries. If this were not so, our method of assembling the test collection would lead to performance figures distorted in favour of program Thomas. We need to know that there is a choice for the program to make, in order to see how discriminating it really is.

Firstly, the supergraph represented in the test data base is a single connected graph; i.e. every point contained in it is reachable from every other point. If we eliminate the check tags, and restrict our consideration to paths of the form:

D —— S —— D —— S —— .... ——D,

where the D's are document nodes and the S's are subject or name nodes, then we find that no document node is more than 8 lines distant from every other document node in the supergraph. It is therefore possible to conduct a dialogue in which a pair of the most widely separated references are

displayed within a few exchanges with the program.

The overlap of topics covered in the test dialogues with Thomas (which are described in the following sections) is shown here in the form of a hierarchical clustering of searches. The searches are those conducted with Thomas, using the "standard rules" described in section 3, below, and corresponding to the 32 productive Medusa searches. For this purpose, a search is defined by the set of references displayed, and the similarity of two searches is measured in terms of the overlap of their defining sets. The technique for producing a hierarchy of clusters is that described by Jardine & van Rijsbergen(1971). We calculate the similarity measure for every pair of searches, A and B:

$$S_{AB} = \frac{|A \cap B|}{|A \cup B|} .$$

For any value of θ, $0 \leqslant \theta \leqslant 1$, we can draw a graph in which the nodes correspond to searches, and a line joins every pair of nodes, A and B, for which $S_{AB} \geqslant \theta$. The clusters at level θ are defined to be the connected components of the graph obtained for that value of θ. We set $\theta = 0$ initially, and draw the graph; then we increase the value of θ. At various values, lines disappear from the graph. If, as a result, a cluster is split into two or more smaller clusters, the value of θ is called a "splitting level". We present the clustering obtained in figure 15: it can be seen that the searches overlap each other to a considerable extent.

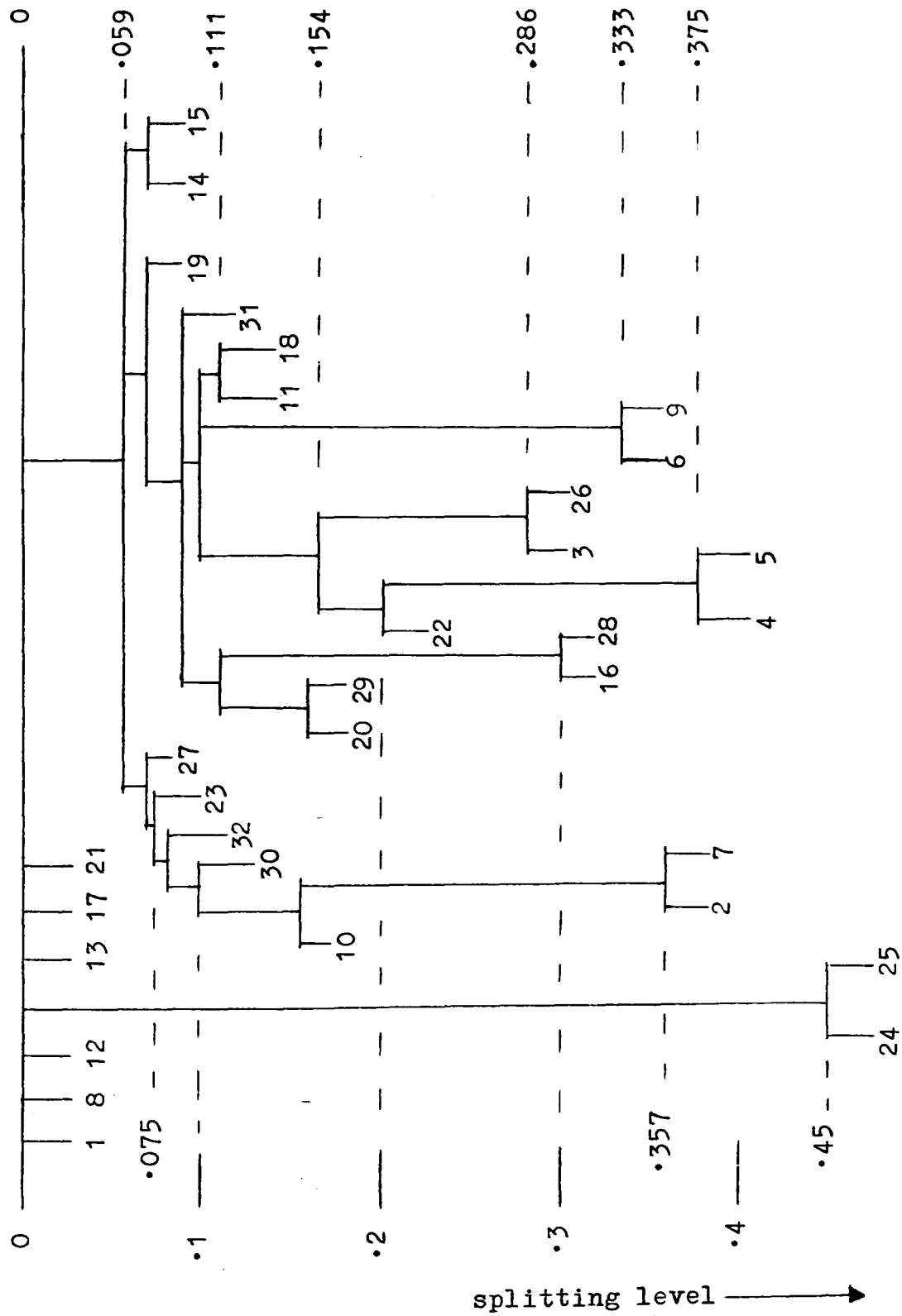


Figure 15. Hierarchical clustering of 32 Thomas dialogues (see text).

3. The trials

The design of the trials for our program was guided by the desire to simulate the behaviour of real users. Salton(1972), in his comparison of SMART and MEDLARS, is prepared to accept the validity of relevance judgements made by subject specialists other than those who posed the original queries. He reports a 69% overlap in relevant sets as judged by users and an independent assessor. The problems of obtaining relevance judgements have been studied by Cuadra et al(1967). However, in testing a system designed for interactive use by the scientist himself, it seems necessary to use his own decisions. The example search given below shows that it is not at all obvious how the user demarcates the output of a retrieval system. Medusa users return relevance judgements to the project team, and those applying to our subset of the collection were available for use in the trial dialogues. All references retrieved by Medusa in response to a search are marked, by the user, in one of four ways:

A : relevant, useful, already known

B : looks relevant, not known, intend to read

* : not relevant but interesting in another connection (serendipity)

\- : not useful

For our present purposes, we regard A and B as meaning "relevant" - the user would respond yes if the reference were displayed by Thomas. The marks * and - are both taken to mean "not relevant" - the response to Thomas is no. This may be a little harsh on our program because, in reality, a user may wish to be non-committal about a refer-

ence, and in that case, a negative response may be misleading. Also, in many cases, the decision as to whether a reference should be marked * or B is difficult to make: one would expect a strong connection between a user's various professional interests. All the analyses given here are derived from dialogues based upon the 32 Medusa searches which yielded relevant references.

The first thing that was necessary, in preparation for the trials, was a summary of each Medusa search, giving an outline of the formulation, including lists of terms selected by the user, and a list of all the references retrieved from the September 1973 section of the file, together with the relevance judgements. In addition, the number of postings in the test collection was noted for each term chosen by the searcher during the Medusa session (that is, for those terms present in the test file). An example will show the form of the summaries prepared (refer to Chapter 2, section 2.4.1, for a sample Medusa session):

Name of search: MWA2 "Adrenal medulla"

Formulation:

entered by user: ADRENAL MEDULLA (m1)
SECRETION (m2)
INNERVATION (q2)
STORAGE (not in dictionary)
CATECHOLAMINES (m3)

thesaurus search from m3,

user chose: DOPA (m4)
DOPAMINE (m5)
EPINEPHRINE (m6)

NOREPINEPHRINE (m10)

entered by user: MORPHINANS (m12)

thesaurus search from m12,

user chose: CODEINE (m13)

DIACETYLMORPHINE (m14)

MORPHINE (m17)

entered by user: NICOTINE (m21)

ATROPINE (m22)

search prescription:

r6 = m1 and (m4 or m5 or m6 or m10 or m13 or m14 or m17 or m21 or m22)

expected return - small

Retrieved references (titles only): Relevance

1. Catecholamine storage in liver metastases of a malignant carotid body tumour. A biochemical and morphological study. B
2. Isolated chromaffin granules maintenance of ATP content during incubation at 31 degrees C. B
3. Urinary epinephrine and norepinephrine responses to chair restraint in the monkey. *
4. Tetrahydroisoquinoline alkaloids: uptake, storage, and secretion by the adrenal medulla and by adrenergic nerves. B
5. Catecholamine response of chickens to exogenous insulin and tolbutamide. B
6. Uptake of calcium in chromaffin granules of bovine adrenal medulla stimulated in vitro. *
7. An analysis of pulse frequency as an adrenergic excitant in pulsatile circulatory support. *

Terms in test file, with postings:

ADRENAL MEDULLA	7
CATECHOLAMINES	5
DOPA	1
DOPAMINE	3
EPINEPHRINE	8
NOREPINEPHRINE	11
DIACETYLMORPHINE	1
MORPHINE	2
ATROPINE	1

The search specification, r6, was arrived at in several stages, interleaved with thesaurus searching and the entry of new terms.

Now we are ready to conduct a search with program Thomas. In the absence of the user, we must have a set of rules to follow during the course of the dialogue:

Rule 1. Start the dialogue by typing the first term entered by the user. If it is not found, try the next, and so on.

Rule 2. In response to references displayed, answer YES for those marked A or B by the user, NO otherwise. Follow this general relevance judgement by a detailed one, if appropriate: terms listed with the reference which were entered or chosen by the user (and are therefore in the Medusa session summary), should be "recognized" the first time they occur.

Rule 3. Stopping rule: stop when all A and B references in the corresponding Medusa search have been displayed.

We shall refer to these rules as the standard rules. As an example we give a dialogue with Thomas, corresponding to the Medusa session summarized above.

	Ref. no. in summary
'ADRENAL MEDULLA'	
Isolated chromaffin granules maintenance of ATP content during incubation at 31 degrees C.; Baumgartner et al, Eur J Pharmacol,22,102-4,Apr 73. 1. H.Baumgartner, 2. H.Winkler, 3. H.Hortnagl, 4. adenosine triphosphate, 5. adrenal glands, 6. adrenal medulla, 7. animal experiments, 8. carbon isotopes, 9. catecholamines, 10. cattle, 11. chromaffin system, 12. in vitro, 13. magnesium, 14. norepinephrine, 15. nucleotides, 16. time factors, 17. tritium	2

YES, 9, 14

(Remark: the terms CATECHOLAMINES and NOREPINEPHRINE have been "recognized" according to rule 2)

Catecholamine storage in liver metastases of a malignant carotid body tumour. A biochemical and morphological study.; Hortnagl et al, Virchows Arch <Zellpathol>,12,330-7,30 Mar 73.
1. H.Hortnagl, 2. H.Hortnagl, 3. A.Propst, 4. H. Schwingshackl, 5. G.Weiser, 6. H.Winkler, 7. adrenal medulla, 8. carotid body tumor, 9. catecholamines, 10. chromogranins, 11. complement fixation tests, 12. cytoplasmic granules, 13. dopamine beta hydroxylase, 14. female, 15. human, 16. liver neoplasms, 17. membranes, 18. microscopy, 19. microscopy, electron, 20. middle age, 21. neoplasm metastasis, 22. norepinephrine

1

YES

Tetrahydroisoquinoline alkaloids: uptake, storage, and secretion by the adrenal medulla and by adrenergic nerves.; Cohen, Ann NY Acad Sci,215, 116-9,30 Apr 73.
1. G.Cohen, 2. adrenal medulla, 3. adrenergic false transmission, 4. alkaloids, 5. animal experiments, 6. dopamine, 7. epinephrine, 8. isoquinolines, 9. neural transmission, 10. norepinephrine, 11. rats, 12. sympathetic nervous system

4

YES, 6, 7

An analysis of pulse frequency as an adrenergic excitant in pulsatile circulatory support.; Harrison et al,Surgery,73,868-74,Jun 73.
1. T.S.Harrison, 2. J.F.Seaton, 3. adrenal medulla, 4. animal experiments, 5. assisted circulation, 6. blood pressure, 7. cardiac output, 8. carotid sinus, 9. dogs, 10. epinephrine, 11. hydrogen-ion concentration, 12. norepinephrine, 13. pulse, 14. vascular resistance

7

NO

(Viral hepatitis and the corresponding antigen); Houston, Med Lab(Stuttg),26,101-4,May 73. 1. R.G.Houston, 2. Australia antigen, 3. complement fixation tests, 4. hemagglutination inhibition tests, 5. hepatitis, homologous serum, 6. human, 7. methods	X

NO

Uptake of calcium in chromaffin granules of bovine adrenal medulla stimulated in vitro.; Serck-Hanssen et al, Biochim Biophys Acta,307,404-14,11 May 73. 1. G.Serck-Hanssen, 2. E.N.Christiansen, 3. acetylcholine, 4. adrenal medulla, 5. animal experiments, 6. biological transport, 7. calcium, 8. cattle, 9. chromaffin system, 10. cytoplasmic granules, 11. EDTA, 12. epinephrine, 13. fluorescence spectrometry, 14. microsomes, 15. norepinephrine, 16. perfusion, 17. proteins, 18. spectrophotometry, atomic	6

NO

Catecholamine response of chickens to exogenous insulin and tolbutamide.; Pittman et al, Comp Biochem Physiol (A),45,141-7,1 May 73. 1. R.P.Pittman, 2. R.L.Hazelwood, 3. adrenal medulla, 4. animal experiments, 5. blood sugar, 6. cattle, 7. chickens, 8. epinephrine, 9. female, 10. fluorescence spectrometry, 11. hypoglycemia, 12. insulin, 13. norepinephrine, 14. time factors, 15. tolbutamide	5

YES (Remark: we have seen all the "relevant" references now)

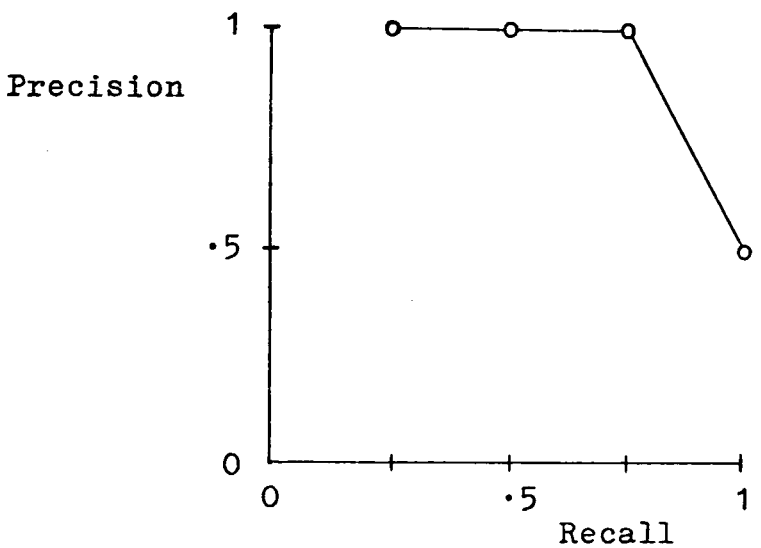
.
.
.

For the purposes of analysis, the search stops with the appearance of reference 5 - "Catecholamine response ...". We shall assume that there are no relevant references in that part of the test file which the user did not see in the Medusa session.

The search can be summarized, for statistical purposes, by the sequence:

R R R - - - R

meaning that we were shown 3 relevant (R) references, followed by 3 non-relevant ones (-) and finally the fourth relevant reference. We have assumed that the recall ratio at the end of the sequence is 1. If we plot precision against recall at each recall level in the sequence, we get a graph like this:



The final precision is 4/7, which happens, in this case, to be the same as that obtained by Medusa. Having fixed certain of the system parameters and the dialogue rules, 32 searches could be run and sequences of R's and -'s found for each one. Graphs corresponding to the sequences exhibit a wide variation and, although we have summarized them below, little weight can be attached to the average performance graphs. Figure 16 contains a few individual performance graphs to show the extent of the variation.

The technique we use for producing the average performance curve from a set such as that in figure 16 is

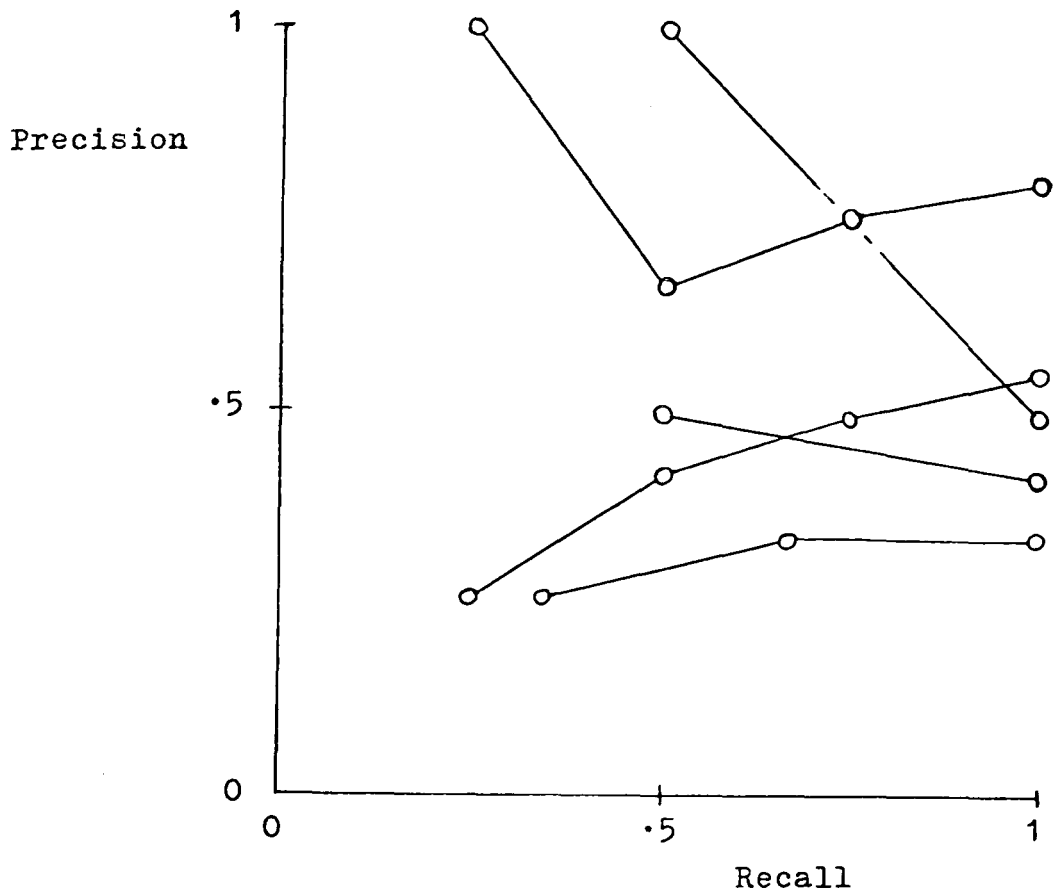


Figure 16. Some individual search performances.

one of the possibilities discussed by Keen(1971) in connection with the SMART system. Firstly, because the curves do not all extend the same distance along the recall axis, we must extrapolate them to the left. We have to decide what the value of precision is before the first reference in a dialogue is displayed. Keen discusses five possibilities and points out that for comparative evaluations, it does not much matter which we choose. We follow the usual practice of assuming that precision is 1 when the recall level is 0. Figure 17 shows a curve so extended on the left. The next step is to standardize the curve. This is necessary because the number of points on

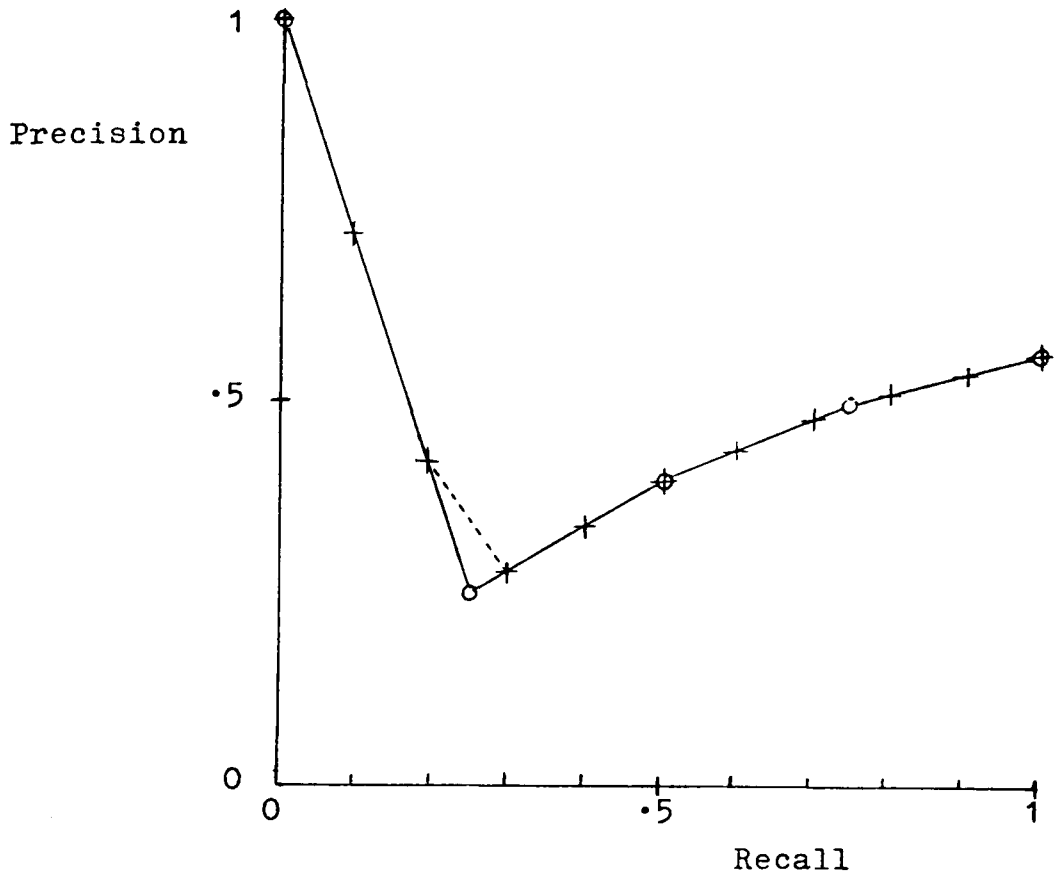


Figure 17. Performance curve extrapolated to the left and standardized.

the curve varies (according to how many relevant documents there are for the query). We choose an increment of recall (say 0·1) and interpolate in all the straight line segments of the curve at the recall values so determined. The points on the standardized curve are shown by crosses in figure 17. Now the curves for a set of searches can be averaged. The result will look like figure 18.

There are two interesting features in these curves. The final precision, π in figure 18, is the average of the final precision ratios of the separate searches and is related, inversely, to the search length in retrieval

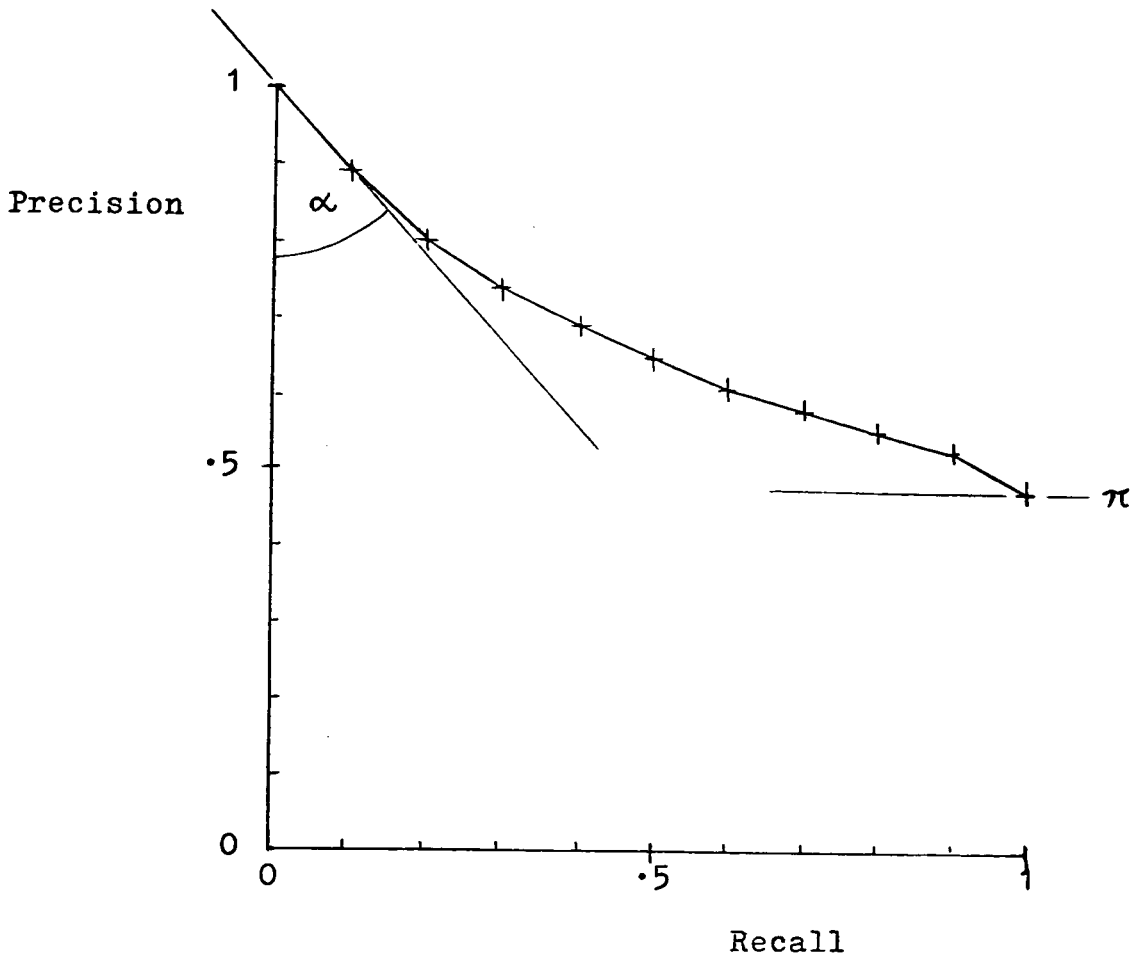


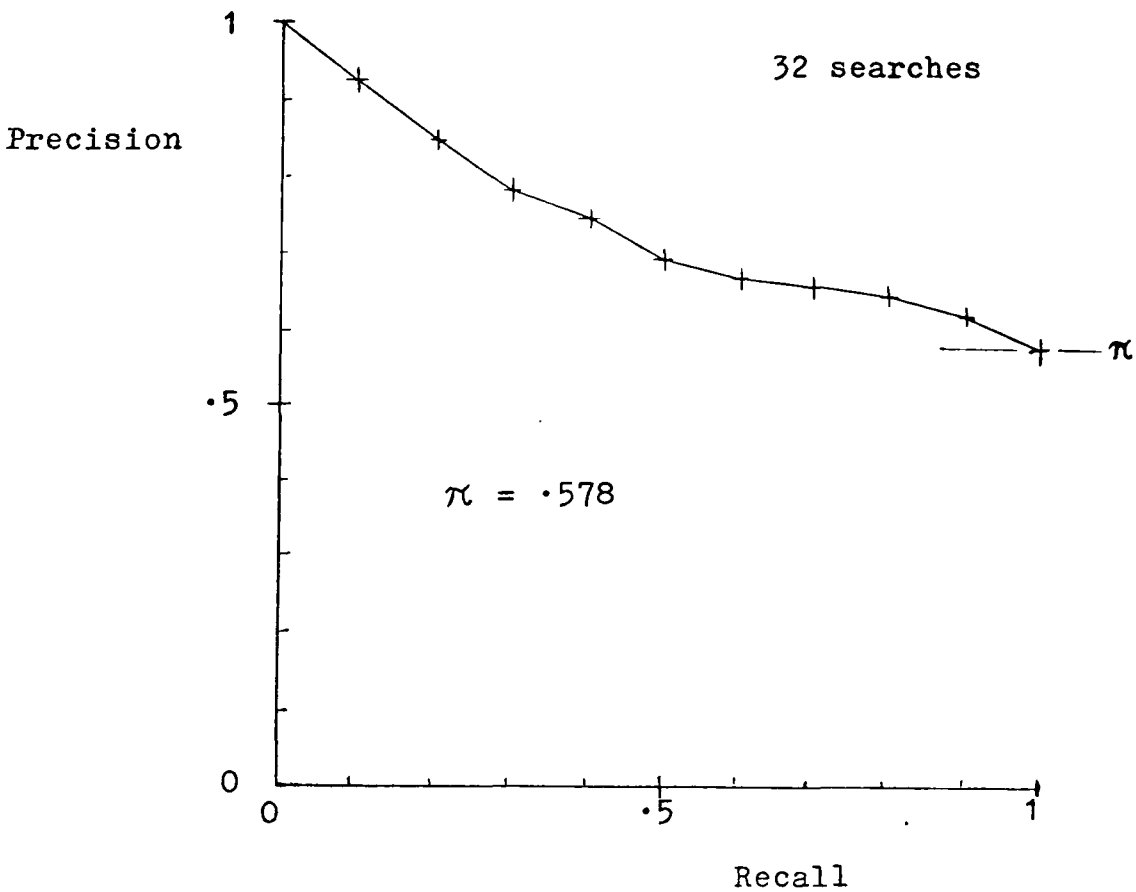
Figure 18. Average performance curve.

dialogues. An average final precision of 1 would mean that all the relevant documents come out before any non--relevant ones appear, for every search; i.e. the search length is always minimum. The slope of the left-most segment of the curve indicates how soon the first relevant document appears. The smaller α is, in figure 18, the longer the system takes to retrieve the first relevant reference. If $\alpha = 90^{o}$, the first reference displayed would always be relevant.

The average performance curve obtained from the 32 dialogues corresponding to fruitful Medusa searches, and

conducted according to the standard rules given above in this section, is shown in figure 19.

Figure 19. Average performance - basic trials, standard rules.



Recall	Precision
0	1
·1	·922
·2	·845
·3	·780
·4	·740
·5	·703
·6	·687
·7	·674
·8	·652
·9	·615
1.0	·578

4. Comparison with Medusa

We cannot produce a compatible average performance curve for the Medusa searches, because that system does not impose an ordering on the retrieved references, so that there are no sensible cutoff points. There are, however, other ways of expressing the performance of the program, which offer crude methods of comparison with Medusa, and are more appropriate to the intentions in the design.

It is claimed of our program that a user can obtain satisfactory performance at low cost in terms of his initiative and effort. The rules of the trials are intended to be usable by an experimenter who is ignorant of medical and biochemical vocabulary. Very little initiative is needed. We should like to compare the efforts expended by the users, and simulated users, in the two sets of dialogues, and also the effectiveness with which the two systems select relevant references. Unfortunately, neither comparison is entirely straightforward.

The nature of a user's effort varies from one part of a dialogue to another. Sometimes he must make selections, sometimes think of words, on other occasions assess relevance; and, of course, there is the physical effort of typing commands or responses. A simple approach has been taken here to obtain a comparison: we just count "tokens" typed in each dialogue. Medusa tokens are:

(i) command names, e.g. COMBINE, UP, SEARCH,

(ii) subject terms: multi-word phrases count as one token,

(iii) system-assigned codes, e.g. M9, R6, Q13; count one each time the user types one,

(iv) logical connectives: AND, OR, NOT, LINK; each count

as one token.

Thomas tokens are:

(i) subject terms,

(ii) special words: YES, NO, NOT,

(iii) numbers, repeated by the user from displays,

(iv) null messages, e.g. no comment about a displayed reference.

Effort estimates, expressed as counts of tokens, are listed in Table 5 for the 32 Medusa searches which we are using (e_M) and for the corresponding dialogues with Thomas, using the standard rules (e_T).

The other comparison we must consider is retrieval effectiveness. Unlike interaction with Medusa and most other reference retrieval systems, a dialogue with Thomas has no query formulation phase. The user should approach the relevant references by viewing, and judging, a sequence of references in the neighbourhood of the nodes in which he has expressed interest. We should not expect the first reference displayed to be relevant; in the early stages, the program will normally have little information to act upon. The characteristics of interest are:

(i) how quickly the program displays the first relevant reference, and

(ii) to what extent its output remains pertinent up to the point when all the relevant references have been displayed.

The first can be measured by counting the number of non-relevant references displayed before the first relevant one. The second by the proportion of relevant documents among those that follow, up to the last relevant one, i.e.

the precision at recall level 1, ignoring leading non--relevant references. In table 5, we list this precision value (π'), the overall precision as defined in the previous section (π), and the number of leading non-relevant references (λ) for each of the 32 Thomas dialogues. If, for example, a dialogue is summarized by the sequence:

- - R - R R - R

the values which go into the table are:

$$\lambda = 2,$$

$$\pi = 4/8 = \cdot 5$$

$$\pi' = 4/6 = \cdot 67$$

For comparison, the table also includes the Medusa precision figures (π_M). The comparison can be made between π and π_M, in which case one makes no allowance for query formulation (or the establishment of the context) in Thomas, or, regarding the leading non-relevant references as equivalent to Medusa query formulation, one compares π' and π_M. An appropriate, powerful statistical test for the overall relationship between the columns, as suggested by the averages, is the Wilcoxon matched-pairs signed-ranks test (Siegel,1956). This is a non-parametric test (no assumptions about the distribution of the data need be made) which takes account of the pairwise difference between two samples. Application of the test tells us that the difference between π and π_M is not statistically significant, and that $\pi' > \pi_M$ is acceptable at the ·01 level of significance. We can be confident that $e_M > e_T$ (in fact, every number in the e_M column exceeds the corresponding one in the e_T column).

We conclude that, for the test collection and under

Search no.	Thomas dialogues, standard rules				Medusa searches	
	λ	π	π'	effort e_T	effort e_M	π_M
1	1	·4	·5	6	18	·4
2	0	·2	·2	14	25	1
3	1	·6	·75	8	16	·6
4	0	1	1	5	13	·5
5	3	·57	1	9	13	·5
6	0	·5	·5	4	28	1
7	0	1	1	11	31	·875
8	0	1	1	4	25	1
9	1	·5	1	2	11	·5
10	1	·33	·5	13	43	·25
11	2	·6	1	16	73	·375
12	2	·5	1	9	48	·4
13	0	·8	·8	9	46	·8
14	0	·625	·625	16	68	1
15	0	·8	·8	8	28	·8
16	0	·71	·71	7	19	·833
17	1	·33	·375	14	29	·2
18	0	1	1	5	46	1
19	0	·83	·83	9	20	·357
20	0	1	1	1	13	1
21	2	·33	1	4	33	·33
22	0	·27	·27	20	33	·375
23	5	·29	1	11	48	·18
24	3	·29	·5	12	52	·22
25	6	·14	1	15	40	·33
26	0	·36	·36	19	56	·57
27	0	1	1	6	19	1
28	1	·5	·67	8	25	·5
29	1	·6	·75	6	19	·75
30	3	·6	·86	15	47	·5
31	0	·67	·67	5	24	·5
32	7	·13	1	14	55	1
Averages:	1·25	·58	·77	9·5	33·25	·61

Table 5. Thomas - Medusa comparison. (See text)

standard dialogue rules, program Thomas is about as effective in retrieval as Medusa, but at lower demand on user effort and without requiring the user to formulate a query. If one wishes to regard the first few interactions (on average 1·25) as equivalent to query formulation, then Thomas is more effective than Medusa - precision is about 25% better in Thomas. As with most small scale experiments, we cannot infer that the performance will be equally satisfactory when Thomas (or a program like it) is operating on a realistic collection of references. Clearly, an important factor in a large file is the generally much higher level of term postings. If a user starts his search with a very highly posted term, of order 1000 postings, for example, the initial sequence of non-relevant references could be rather long. It is difficult to issue guidance to the user, such as "try to avoid broad terms", because his impression of term specificity will not always be in accord with statistical specificity, which is what matters. In our file, for instance, the term CATECHOLAMINES is adjacent to 5 reference nodes; it is more specific, statistically, than either NOREPINEPHRINE (11 references) or EPINEPHRINE (8 references), both of which are narrower terms - i.e. lower in the thesaurus hierarchy (MeSH) and likely to be thought of by the user as specific. It will be recalled that the program monitors its performance (Chapter 4, section 3.2.1) and, if it appears to be doing badly, will return to a reference that the user has approved of, or show him subjects related to one that he has selected (Chapter 4, section 3.3.3). It is at this point that suggestions could be made to him, based upon

term postings. Some tests have been done to gauge the variations in performance due to initiating a dialogue with terms of different specificity, and these are described in the next section. The samples are small, and once again no guarantee can be given that the results would be reflected in large scale searches.

5. Further experiments

Three sets of trials have been made to determine the program's performance under conditions of minimum effort by the user, and also to show the effect of the specificity of the starting point in the dialogue upon the search length. The dialogue rules are as follows:

Rule 1. Start the dialogue by typing one term: the rule for choosing this term is discussed below.

Rule 2. In response to references displayed, answer YES for those marked A or B by the user, NO otherwise. Note that we do not "recognize" any displayed terms.

Rule 3. Abort search if the program requests the user to supply another term. The program does this if there are no documents in its model that the user has not already seen, i.e. it is stuck.

Rule 4. Normal stopping rule: stop when all A and B references in the corresponding Medusa search have been displayed.

Now we define a family of three sets of dialogue rules by varying the criterion for selecting the starting term (Rule 1). The selection must always be made from those terms typed in, or selected by the Medusa searcher.

Using the same example as we used in section 3, i.e. the search entitled MWA2 "Adrenal medulla", the candidate terms are:

term	postings in test collection
ADRENAL MEDULLA	7
CATECHOLAMINES	5
DOPA	1
DOPAMINE	3
EPINEPHRINE	8
NOREPINEPHRINE	11
DIACETYLMORPHINE	1
MORPHINE	2
ATROPINE	1

The dialogue trials:

(i) High posting rules. Start with the term with highest posting - NOREPINEPHRINE in the example. The term must be associated with at least 9 references in the entire collection, and it must not be a check tag (see section 2).

(ii) Medium posting rules. Use a term with a posting number in the range 2 to 10, wnich is closest to the average for terms in the list for the search under consideration. In the example it would be CATECHOLAMINES.

(iii) Low posting rules. Use the term with lowest posting (must be 1 or 2). In the example we would use ATROPINE (in preference to DOPA and DIACETYLMORPHINE, because the original user typed it, rather than chose it from a thesaurus display).

The values obtained for the number of leading non-relevant references, the overall precision, and the precision from the first relevant reference are recorded in table 6. Those resulting from the standard rules are repeated

(λ, π, π'), and the values obtained using the high posting rules $(\lambda_h, \pi_h, \pi_h')$, medium posting rules $(\lambda_m, \pi_m, \pi_m')$ and low posting rules $(\lambda_l, \pi_l, \pi_l')$ are included. It is not always possible to apply all three variants of the rules to a search, so the number of observations in the experiment is reduced. In addition, some searches had to be aborted (Rule 3):

High posting rules - 2 aborted,

Medium posting rules - 4 aborted,

Low posting rules - 11 aborted.

This was mainly due to the absence of relevant documents described by the chosen terms, and the high figure (11) for the low posting rules serves to emphasize the disparity between statistically specific words (i.e. ones that are used little) and conceptually specific words.

Because the performance of a retrieval system depends very much upon the query, statistical tests to estimate the significance of the differences between average performance figures must be based on the differences between matched pairs of measures. For this reason, we cannot obtain samples from table 6 which are large enough to give statistically significant results. This being said, however, the table does tend to confirm our expectations of the program's behaviour. The search length increases with the number of postings of the term that the user starts with. Not only is the length of search before reaching a relevant reference larger for highly posted terms, but the continuing search appears to be longer (indicated by the lower precision, π_h'). The program's model of the user's interest starts by being large, and

Search no.	Standard rules			High posting rules			Medium posting rules			Low posting rules		
	λ	π	π'	λ_h	π_h	π'_h	λ_m	π_m	π'_m	λ_l	π_l	π'_l
1	1	.4	.5	-	-	-	1	.4	.5	-	-	-
2	0	.2	.2	2	.29	.4	8	.15	.4	0	.4	.4
3	1	.6	.75	1	.6	.75	0	.5	.5	-	-	-
4	0	1	1	0	.67	.67	0	.4	.4	0	.5	.5
5	3	.57	1	3	.4	.57	-	-	-	-	-	-
6	0	.5	.5	3	.29	.5	-	-	-	0	.5	.5
7	0	1	1	0	1	1	-	-	-	0	1	1
8	0	1	1	-	-	-	0	1	1	-	-	-
9	1	.5	1	8	.11	1	1	.5	1	-	-	-
10	1	.33	.5	1	.2	.21	3	-	-	-	-	-
11	2	.6	1	1	.25	.28	0	.75	.75	0	1	1
12	2	.5	1	-	-	-	0	1	1	-	-	-
13	0	.8	.8	-	-	-	0	.8	.8	-	-	-
14	0	.625	.625	-	-	-	0	.625	.625	-	-	-
15	0	.8	.8	-	-	-	0	.8	.8	0	1	1
16	0	.71	.71	-	-	-	0	.71	.71	-	-	-
17	1	.33	.375	0	.21	.21	-	-	-	-	-	-
18	0	1	1	-	-	-	0	.6	.6	0	.6	.6
19	0	.83	.83	1	.45	.5	0	.32	.32	-	-	-
20	0	1	1	-	-	-	-	-	-	0	1	1
21	2	.33	1	2	-	-	-	-	-	-	-	-
22	0	.27	.27	0	.27	.27	-	-	-	-	-	-
23	5	.29	1	4	.33	1	-	-	-	-	-	-
24	3	.29	.5	-	-	-	3	.33	.67	-	-	-
25	6	.14	1	3	.25	1	-	-	-	-	-	-
26	0	.36	.36	0	.36	.36	3	.4	.57	-	-	-
27	0	1	1	-	-	-	0	1	1	0	1	1
28	1	.5	.67	-	-	-	3	.33	.67	-	-	-
29	1	.6	.75	-	-	-	1	.6	.75	-	-	-
30	3	.6	.86	-	-	-	4	.55	.86	0	.75	.75
31	0	.67	.67	-	-	-	0	.4	.4	0	.5	.5
32	7	.13	1	-	-	-	2	.33	1	-	-	-
Averages	1.25	.58	.77	1.81	.38	.58	1.26	.57	.70	0	.75	.75
No. in sample	32			16	15		23	22		11		

Table 6. Specificity tests data. (See text).

must be reduced: the refinement continues after the first relevant reference is displayed. We can also see from the table that the system's performance is acceptable, even when the user makes very little effort indeed. Altogether, 63% of the searches conducted under these rules succeeded in achieving 100% recall (i.e. were not aborted).

Clearly, with a collection of realistic size, a user who contributes little initiative is likely to encounter long searches because the terms (in a controlled vocabulary such as we have used) will frequently be highly posted in comparison to the norm in this test collection. If, however, the user is able to give more direction to the search by making more detailed responses to the program's displays, the search length will then depend upon the distance in the network from his starting point to a relevant document node. The test collection generates a network which is not only connected, but highly convoluted, i.e. no node is very far from every other node (section 2). To prove the same for a large collection is a formidable task; but it seems very likely to be true. The subject terms (MeSH) are arranged in several trees, known as categories. The maximum depth of the trees is four nodes, so the path from one leaf node to another in the same category is at most 6 lines long. There are three means of moving from one category to another, in very few steps: (i) many terms belong to more than one category (no steps), (ii) there are a large number of cross references between terms in different categories (1 step), (iii) most documents are associated with terms from different categories (2 steps). It would be very surprising if any

category of terms were completely cut off from the rest. Finally, if there is a path between every pair of subject nodes, then there is a path between any two document nodes.

Chapter 8

CONCLUDING REMARKS

There are several topics to which further work could be devoted: improvements to the information heuristics and data recognition algorithms, studies of file handling for large, rich networks of records, indexing languages with more complicated features (involving syntax) and their use within an on-line program of the Thomas sort, investigation of problems arising out of file size. We feel that the last of these is sufficiently important to warrant a discussion before we conclude. Finally, we reiterate the main theme of this work.

1. The problem of scale

It is important, in the reference retrieval field, that the experimenter who chooses, or is constrained, to use a small collection of documents should bear in mind the applicability of his designs to realistic, large scale collections. Can we predict the performance of Thomas, or a "production model" based on Thomas, operating on a data base concerning a useful field-specific collection of, say, 100,000 documents? The problem is not simply one of implementation; we should like to know whether the heuristics used by Thomas will still be able to help the problem-solving, browsing user. In fact, we believe that the two aspects of the difficulty are closely related. Both the man and the machine will run into trouble if associations become too numerous in the network.

We shall assume that the supergraph (see Chapter 4)

will consist of data similar to that used in the experiment discussed in Chapter 7. The document nodes are associated with author nodes and with subject nodes corresponding to the terms assigned to them from a controlled vocabulary by the indexers. Links between subject nodes are derived from the indexing language thesaurus. Under these circumstances, the major cause of difficulty will be the highly posted terms, which will generate very large stars in the supergraph centred on certain subject nodes and containing several thousands of document nodes. If such clusters are brought into the context graph, the program's model would become unusable. Not only would each choice of a reference for display be a major task for the central processor, but the likelihood of the choice being a successful one would be greatly reduced.

We must concentrate on the size of the model of the user's interest. In a full-scale operational system, there must be features which restrict its size: we cannot allow the model to grow in proportion to the size of the supergraph. Let us examine the components of the model, as listed in section 2 of Chapter 4, and see how points are added to these during a dialogue. We start with the straightforward sets of points:

(i) "explicit requests", "good documents", "accepted documents" and "reviewed nodes" (see Chapter 4 for their definitions) are all limited in size by the length of the dialogue.

(ii) "last selected": the points selected by the user from the last display - clearly a small set.

(iii) "inhibit list": the points rejected by the user, throughout the dialogue. This set grows as the dialogue progresses, but every point in it must have been involved in at least one display so, once again, the length of the dialogue is the limiting factor.

(iv) "context graph". Points are added by procedures ADD_TO_CONTEXT (Chapter 4, section 3.2.3) and FIND_NODES (Chapter 4, section 3.2.4). These include a few specified, individually, by the user, but the main bulk of points will normally be supplied by the function procedures LINKED_DOCUMENTS (called by ADD_TO_CONTEXT) and STARS (called by FIND_NODES). Both of these functions retrieve the sets of points adjacent to those in their arguments. (The procedure UNIFY_CONTEXT_GRAPH - Chapter 4, section 3.2.5 - also causes these functions to be invoked on occasion).

Now, the basis of a constraining feature is already present in the program - the treatment given to the built-in set of check tags. These have cropped up several times in the thesis, and were first mentioned in Chapter 3, section 3.1.1. They are highly posted subject terms which were given special status in Thomas because it was found, in the course of development, that performance was seriously impaired unless their use was restricted. The form that the restriction took was simply to prevent, at all times, LINKED_DOCUMENTS and STARS from fetching the neighbours of any check tag nodes from the supergraph. The effect of doing this is that no document node can be considered for display solely on the basis of the presence of a check tag, even if the user explicitly shows interest in that term. If, however, a document is already in the context graph,

any user-selected check tags with which it is associated will count in its favour when the program makes a choice for display.

Check tags are defined, a priori, for the Medlars indexers, and are consequently among the more highly posted terms. There are other terms which are used very frequently; and if the system were applied to another file of indexed references, there may very well be no equivalent to the "check tag" list in Medlars. We need more flexibility than is provided by a prescribed set of check tags. Many measurements have been taken in the past few years on the use of indexing terms, and there is remarkable consistency among the various indexes: terms are posted according to a hyperbolic distribution, i.e. a few terms are used very frequently, and the frequency figure falls rapidly so that many terms are used only rarely (Fairthorne,1969). Houston & Wall(1964) studied term frequencies found in ten indexes and fit the observations to a family of log-normal distributions (the points fit hyperbolic distributions just as well). Figure 20 shows the cumulative distribution of postings given by Houston and Wall for a collection of 195,000 references. We propose that, for Thomas' successor, terms be added to the check tag set when their frequency of use reaches some chosen level, say 400 postings. Using figure 20 as an example, approximately 15% of terms would then be restricted and about 65% of all terms would have fewer than 100 postings.

On the question of implementation, we note that the restricted subject nodes are distinguished from the others by the fact that the program never wants to know their

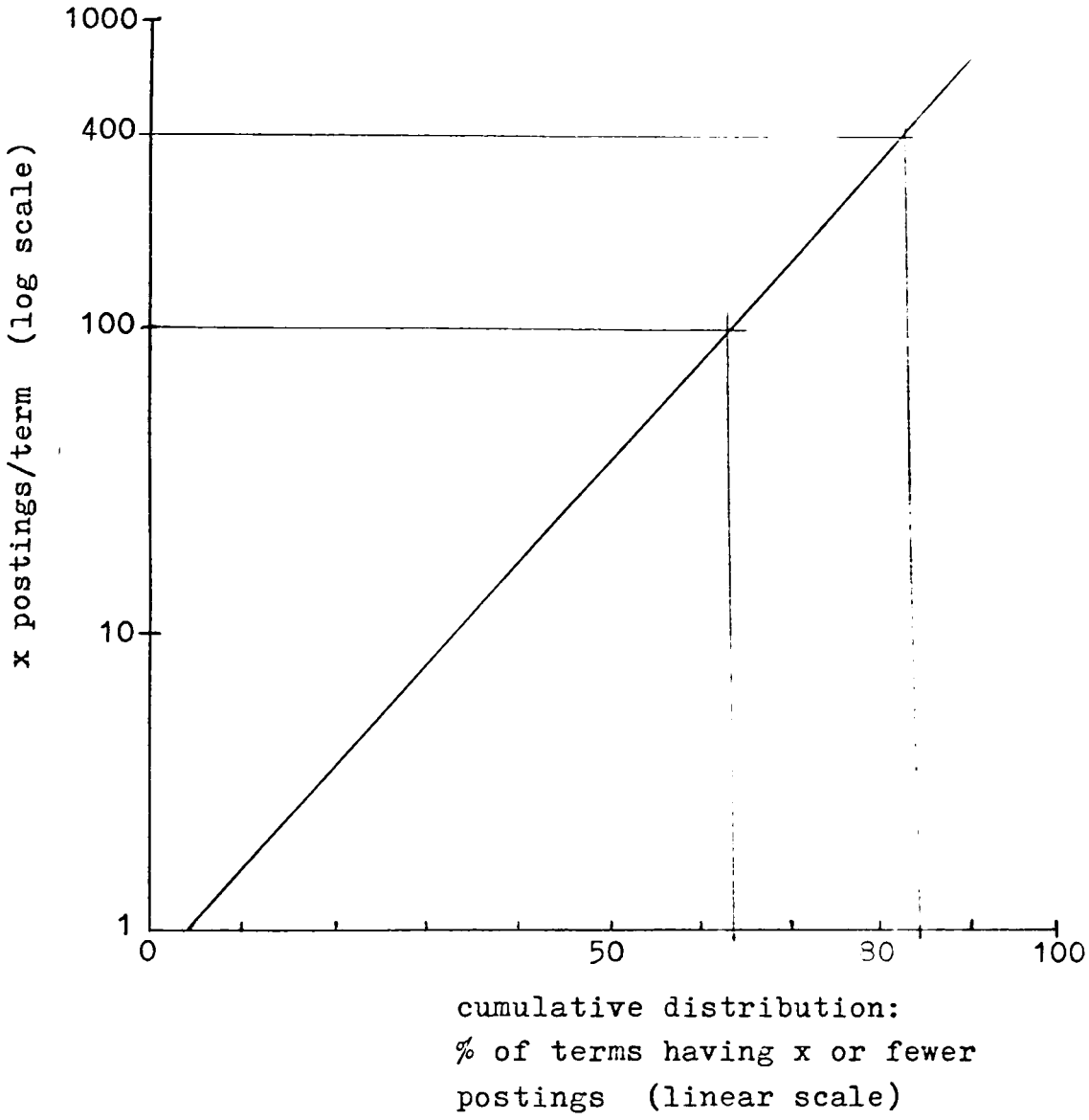


Figure 20. Cumulative distribution of term postings to 195,000 documents. (Houston & Wall,1964). The slope and intersect at x = 1 vary from one index to another.

immediate neighbours in the supergraph. We can, therefore, simultaneously do away with explicit mention of check tags in the program, and all the long lists of file addresses which represent the links from restricted subjects to documents. The supergraph is now a directed graph in which if there is a line from point x to point y, then there is also a line from y to x, unless x is a document point and y a restricted subject point. When the record of a restricted subject is retrieved, no links to documents will be available, although links to other subjects may be. Removing unwanted pointers, and marking the record as belonging to the restricted class (so that links are not added to it in future) can be easily carried out by the network updating procedure as soon as the number of postings exceeds the limit.

In the present project, we have developed a program to implement a particular type of mechanical assistant for the browser. It is far from being a complete reference retrieval system: there are no convenient aids for the (librarian) manager of the system, for example. Problems of size should be considered in the context of a complete system. Decisions on such matters as the indexing procedure and the collection weeding methods should be tackled in the light of file size. Hence the comments above should be regarded merely as guide lines, and an argument in favour of the feasibility of our technique for retrieval applied to real-life document collections.

Before we leave the subject of size, we should emphasize that, although the proposal to severely restrict the role of highly posted terms would seem to lead to

practicable information handling, we have no reliable evidence that it will not impair the retrieval capability of the program, when applied on a large scale. In Thomas, 28 (2·06%) of the 1357 MeSH terms were designated check tags; we were forced to distinguish them in order to obtain the sort of dialogues we had in mind. On the other hand, with a large network, we may wish to restrict as many as 1500 (15%) of the 10,000 terms in MeSH to eliminate the large stars. Can we be sure that the information lost is not significant for our purposes?

There is some evidence that medium and low posted terms are more useful for retrieval than high posted ones; in small experimental collections, at least. Sparck Jones (1972b; 1973c) has discussed, and established the utility of, term weighting based on the frequency of occurrence of the term in the document collection. The weight of a term, and therefore its influence in linear associative retrieval, varies inversely as the number of times the term is posted. There are other ways of weighting index terms (see Sparck Jones,1973c), but the use of collection frequency weighting gives the most notable improvements in performance over unweighted index terms, in small-scale experiments. In other words, if we reduce the influence of highly posted terms in relation to that of less frequently used terms, the retrieval mechanism becomes more effective. The experiments of Salton & Yang(1973) and Svenonius(1972) appear to confirm this, and it is assumed to be true on a large scale by Williams(1969), who has incorporated collection frequency term weights, called "information values", in the BROWSER system.

2. A summing-up

The problem of reference retrieval can be said to be one of communication: the transference to and interpretation by an information system, of incompletely formed ideas. The source of the ideas is the searcher, and we take the view that searching is part of his problem-solving activities. He must at times use the information system in the process of completing his ideas. The program design described and discussed in the foregoing chapters acknowledges the fact that under these circumstances a user cannot easily specify his requirement, even in his own language. If we permit the enquiry to be made in natural language, we are faced with an interpretation problem which, as yet, has no satisfactory solution, save within a very tiny universe of discourse. If we simply extract keywords from the user's question, or ask him to put the question in a simple, rigid form using keywords, then we must have ways of locating instances of the same or closely related concepts, expressed by means of different words.

We have here the ideal application area for interactive computing. A search cannot usually be entirely delegated to a machine: it is really a problem to be solved by the man who, because the task contains so much tedium, can be aided by a computer. The retrieval system should be a synthesis of man and machine. On-line information retrieval systems are now plentiful, and the main reason for their success is the possibilities they offer for interleaving human decisions and mechanical processes.

In spite of all the effort that has been expended on these systems, however, we thought that for many searches

query formulation is still difficult and unreliable, and decided to try to do without it. The program Thomas allows a user to browse among the references (in preference to the term thesaurus) in its data base and, because the normal mode of operation is to show the user references, one after another, there is no need for a "command language", as found in most on-line systems. The user must, of course, specify at least one topic of interest; but during most of his dialogue with the machine he will be deciding upon the pertinence of what he is shown by the program. The computer's actions will be determined by the decisions which he makes known to it. Thus, the man is brought into the system, doing the task for which he is particularly suited.

In program Thomas, the dialogue structure is new. The conceptual basis for it is the model, or representation, that a person develops during conversation of another's view of the world. He does this so that he may understand the ideas being conveyed: the meanings of words depend very much upon the context in which they occur, and the knowledge against which they are set by both the speaker and the listener. Our program uses the enquirer's messages to build up a picture of the bibliographic context of his problem. The program's displays should help the user to appreciate how the words have been used to describe the documents, so that the man also constructs a picture of the other's - the program's - "viewpoint".

Thomas' knowledge consists of the names of a set of bibliographic items and a large number of associations between them. The model of the user's topic that it builds

is in terms of that; it is a small part of that network of data - a cluster of inter-related items of bibliographic interest: references, subjects and authors' names. The cluster changes as the dialogue with the user progresses, and it is not one of a number of clusters determined statistically from the characteristics of the collection as a whole and independent of any queries. The clusters formed by Thomas to model the searcher's area of interest are dynamic and user-induced as opposed to collection--induced.

Measuring the performance of a system with the objectives that we have set ourselves is not at all straightforward. Firstly, whatever measurements we take on a small test data base may work out rather differently on a large file. Secondly, it is not clear how we should measure the success of a browse. Thirdly, we should observe many users with real information needs. In this project, we have had to restrict ourselves to making sure that the program finds the relevant references in its data base, quickly and without a great deal of effort expended by the user. As a result of the tests carried out, we can say that, on a small collection, Thomas retrieved relevant references about as effectively as the Medusa system (which was the source of Thomas' data base), but the demand on user effort by Thomas is much less than that demanded by Medusa. Thomas achieved this performance without giving the user the ability to formulate a query. If we equate the query formulation phase of a Medusa session with the first few interactions with Thomas, establishing the model, the tests show that Thomas' performance is substantially

better than Medusa's.

The most important follow-up to this work would be the creation of a much larger network, with a suitably modified program. Experiments should be done with many different users, having genuine information needs. It is important that the data available to the program should be sufficient to satisfy many of the users' requirements. An enlarged system could be the vehicle for experiments on size, the efficient organization of large graph structures on disk storage, and measurements of the performance of man and computer together, looking for relevant references.

BIBLIOGRAPHY

The names of some of the journals frequently referred to are abbreviated in the following pages. They are:

Am.Doc	American Documentation
CACM	Communications of the Association for Computing Machinery
ISR	Information Storage and Retrieval
JACM	Journal of the Association for Computing Machinery
JASIS	Journal of the American Society for Information Science
J.Doc	Journal of Documentation
JOLA	Journal of Library Automation

* * *

Alberga C.N.(1967). "String similarity and misspellings" CACM,10, pp302-13.

Augustson J.G. & J.Minker(1970). "Deriving term relations for a corpus by graph theoretical clusters" JASIS,21, pp101-11.

Austin D.(1974). "The development of PRECIS: a theoretical and technical history" J.Doc,30, pp47-102.

Ayres F.H., J.German, N.Loukes & R.H.Searle(1968). "Author versus Title: a comparative survey of the accuracy of information which the user brings to the library catalogue" J.Doc,24, pp266-72.

Bar-Hillel Y.(1964). Language and information: selected essays on their theory and application. Addison-Wesley: 1964.

Barraclough E.D., A.S.Barber & W.A.Gray(1972). Medlars on-line search formulation and indexing. University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no.34: 1972.

Barber A.S., E.D.Barraclough & W.A.Gray(1973). "On-line information retrieval as a scientist's tool" ISR,8, pp429-40.

Barker F.H., D.C.Veal & B.K.Wyatt(1972). "Towards automatic profile construction" J.Doc,28, pp44-55.

Batten W.E.(1947). "A punched card system of indexing to meet special requirements" Report of the 22nd Aslib Conference, pp37-9.

Bays C.(1973). "The reallocation of hash-coded tables" CACM,16, pp11-4.

Belkin N.J.(1974). "Towards a definition of information for informatics" Paper presented at Informatics 2, Aslib Co-ordinate Indexing Group Annual Conference, 25-27 March 1974, Oxford.

Blair C.R.(1960). "A program for correcting spelling errors" Information and Control,3, pp60-7.

Bobrow D.G., J.B.Fraser & M.R.Quillian(1967a). "Automated language processing" Annual Review of Information Science and Technology,2, pp161-86.

Bobrow D.G. & D.L.Murphy(1967b). "Structure of a LISP system using two-level storage" CACM,10, pp155-9.

Bobrow D.G. & B.Raphael(1974). "New programming languages for artificial intelligence research" Computing Surveys,6, pp153-74.

Bookstein A.(1972). "Double hashing" JASIS,23, pp402-5.

Borko H. & M.Bernick(1963). "Automatic document classification" JACM,10, pp151-62.

Borko H. & M.Bernick(1964). "Automatic document classification. Part II. Additional experiments" JACM,11, pp138-51.

Bourne C.P. & D.Ford(1961). "A study of methods for systematically abbreviating English words and names" JACM,8, pp538-52.

British Standards Institution(1963). Guide to the Universal Decimal Classification (UDC) B.S.1000C: 1963.

Brookes B.C.(1974) "Robert Fairthorne and the scope of Information Science" J.Doc,30, pp139-52.

Buchholz W.(1963). "File organization and addressing" IBM Systems Journal,2, pp86-111.

Burnaugh H.P.(1967). "The BOLD (Bibliographic On-Line Display) System" Information retrieval: a critical view G.Schechter (Ed.), Thompson: 1967, pp53-66.

Bush V.(1945). "As we may think" Atlantic Monthly,176, pp101-8.

Carville M., L.D.Higgins & F.J.Smith(1971). "Interactive reference retrieval in large files" ISR,7, pp205-10.

Cleverdon C.W.(1972). "On the relationship of recall and precision" J.Doc,28, pp195-201.

Cleverdon C.W.(1974). "User evaluation of information retrieval systems" J.Doc,30, pp170-80.

Cleverdon C.W., J.Mills & M.Keen(1966). Factors determining the performance of indexing systems. Aslib Cranfield Project, Cranfield: 1966. Vol.1: Design (Parts 1 and 2). Vol.2: Test results. (Cleverdon & Keen).

Coates E.J.(1973). "Some properties of relationships in the structure of indexing languages" J.Doc,29, pp390-404.

Cooper W.S.(1973). "On selecting a measure of retrieval effectiveness" JASIS,24, pp87-100; "Part II. Implementation of the philosophy" JASIS,24, pp413-24.

Crouch D.B.(1973). "A process for reducing cluster representation and retrieval costs" Proc. ACM annual conference, 1973, pp224-7.

Cuadra C.A., R.V.Katter, E.H.Holmes & E.M.Wallace(1967). Experimental studies of relevance judgements: final report. June 1967. TM-3520/001/00, SDC, Santa Monica, Calif.

Dahl O.-J., E.W.Dijkstra & C.A.R.Hoare(1972a). Structured programming. Academic Press: 1972.

Dahl O.-J. & C.A.R.Hoare(1972b). "Hierarchical program structures" in Dahl et al(1972a) q.v., pp175-220.

van Dam A. & D.E.Rice(1970). "Computers and publishing: writing, editing, and printing" Advances in Computers, 10, pp145-74.

Dijkstra E.W.(1962). A primer of Algol 60 programming. Academic Press: 1962.

Dijkstra E.W.(1972). "Notes on structured programming" in Dahl et al(1972a) q.v., pp1-82.

Dodd G.G.(1969). "Elements of data management systems" Computing Surveys,1, pp117-33.

Dolby J.L.(1970). "An algorithm for variable-length proper-name compression" JOLA,3, pp257-75.

Dolby J.L.(1971). "Programming languages in mechanized documentation" J.Doc,27, pp136-55.

Doyle L.B.(1961). "Semantic road maps for literature searchers" JACM,8, pp553-78.

Doyle L.B.(1965). "Expanding the editing function in language data processing" CACM,8, pp238-43.

Earley J.(1971). "Toward an understanding of data structures" CACM,14, pp617-27.

Elcock E.W., J.M.Foster, P.M.D.Gray, J.J.McGregor & A.M.Murray(1971). "Abset, a programming language based on sets: motivation and examples" Machine intelligence,6 B.Meltzer & D.Michie (Eds.), Edin. Univ. Press: 1971, pp467-92.

Englebart D.C., R.W.Watson & J.C.Norton(1973). "The augmented knowledge workshop" Proc. National Computer Conference, 1973, pp9-21.

Fairthorne R.A.(1956). "The patterns of retrieval" Am. Doc,7, pp65-70. Reprinted in Fairthorne(1961), q.v.

Fairthorne R.A.(1958). "Delegation of classification" Am.Doc,9, pp159-64. Reprinted in Fairthorne(1961), q.v.

Fairthorne R.A.(1961). Towards information retrieval. Butterworths: 1961.

Fairthorne R.A.(1964). "Basic parameters of retrieval tests" American Documentation Institute, Parameters of information science: Proc. 27th annual meeting. Spartan Books: 1964, pp343-5.

Fairthorne R.A.(1969). "Empirical hyperbolic distributions (Bradford - Zipf - Mandelbrot) for bibliometric description and prediction" J.Doc,25, pp319-43.

Farber D.J., R.E.Griswold & I.P.Polansky(1964). "SNOBOL, a string manipulation language" JACM,11, pp21-30.

Farradane J.(1967). "Concept organization for information retrieval" ISR,3, pp297-312.

Farradane J., J.M.Russell & P.A.Yates-Mercer(1973). "Problems in information retrieval: logical jumps in the expression of information" ISR,9, pp65-77.

Foskett A.C.(1971). The subject approach to information. 2nd Ed., Clive Bingley: 1971.

Foskett D.J.(1972). "A note on the concept of 'relevance'" ISR,8, pp77-8.

Glantz R.S.(1970). "SHOEBOX - A personal file handling system for textual data" Proc. AFIPS,37, pp535-45.

Goodliffe E.C. & S.J.Hayler(1974). "On-line information retrieval: some comments on the use of Retrospec I in an industrial library" Aslib Proceedings,26,pp177-88.

Gorman M. & J.E.Linford(1971). Description of the BNB/MARC record - a manual of practice. BNB/MARC Documentation Service - publication no.5. Council of the British National Bibliography: 1971.

Gotlieb C.C. & S.Kumar(1968). "Semantic clustering of index terms" JACM,15, pp493-513.

Gray W.A. & A.J.Harley(1971). "Computer assisted indexing" ISR,7, pp167-74.

Halmos P.R.(1960). Naive set theory. Van Nostrand Reinhold Company: 1960.

Harary F.(1969). Graph theory. Addison-Wesley: 1969.

Henderson P. & R.A.Snowdon(1972). "An experiment in structured programming" BIT,12, pp38-53.

Herner S.(1970). "Browsing" Encyclopedia of library and information science. Marcel Dekker: 1970.

Heyting A.(1956). Intuitionism: an introduction. North Holland Publishing Co.: 1956.

Higgins L.D. & F.J.Smith(1969). "On-line subject indexing and retrieval" Program,3, pp147-56.

Higgins L.D. & F.J.Smith(1971). "Disc access algorithms" Computer Journal,14, pp249-53.

Hillman D.J.(1964). "Two models for retrieval system design" Am.Doc,15, pp217-25.

Hillman D.J.(1968). "Negotiation of inquiries in an on-line retrieval system" ISR,4, pp219-38.

Hillman D.J.(1973). "Customized user services via interactions with LEADERMART" ISR,9, pp587-96.

Hoare C.A.R.(1972). "Notes on data structuring" in Dahl et al(1972a) q.v., pp83-174.

Houston N. & E.Wall(1964). "The distribution of term usage in manipulative indexes" Am.Doc,15, pp105-14.

IBM System/360 Component descriptions - 2314 direct access storage facility and 2844 auxiliary storage control. Form A26-3599.

Ide E. & G.Salton(1971). "Interactive search strategies and dynamic file organization in information retrieval" Chapter 18 in Salton(1971) q.v., pp373-93.

Informatics 1. Proceedings of a conference held by the Aslib Co-ordinate Indexing Group on 11-13 April 1973 at Durham University. Aslib: 1974.

Jacquesson A. & W.D.Schieber(1973). "Term association analysis on a large file of bibliographic data, using a highly-controlled indexing vocabulary" ISR,9, pp85-94.

Jardine N. & C.J.van Rijsbergen(1971). "The use of hierarchic clustering in information retrieval" ISR, 7, pp217-40.

Johnson R.L.(1974). "An extended ALGOL for language processing" Informatics 1 q.v., pp182-93.

Jones K.P.(1971). "Basic structures for thesaural systems" Aslib Proceedings,23, pp577-90.

Jones P.E.(1965). "Historical foundations of research on statistical association techniques for mechanized documentation" Statistical association methods for mechanized documentation. Symposium proceedings, Washington 1964. Stevens, Guiliano & Heilprin (Eds), pp3-8.

Katz J.J. & J.A.Fodor(1963). "The structure of a semantic theory" Language,39, pp170-210.

Kay M. & K.Sparck Jones(1971). "Automatic language processing" Annual Review of Information Science and Technology,6, pp141-66.

Keen E.M.(1971). "Evaluation parameters" Chapter 5 in Salton(1971) q.v., pp74-111.

Keen E.M.(1973). "The Aberystwyth-index languages test" J.Doc,29, pp1-35.

Kemp D.A.(1974). "Relevance, pertinence and information system development" ISR,10, pp37-47.

Kilgour F.G., P.L.Long & E.B.Leiderman(1970). "Retrieval of bibliographic entries from a name-title catalog by use of truncated search keys" Proc.ASIS,7, pp79-82.

Knuth D.E.(1973). The art of computer programming. Vol.3: Searching and sorting. Addison-Wesley: 1973.

Kraft D.H.(1973). "A decision theory view of the information retrieval situation: an operations research approach" JASIS,24, pp368-76.

Kuno S.(1966). "Computer analysis of natural languages" Symposium on mathematical aspects of computer science; American Mathematical Society, New York, April 1966, pp52-110.

Kunz W. & H.W.J.Rittel(1972). "Information science: on the structure of its problems" ISR,8, pp95-8.

Lancaster F.W.(1968). Information retrieval systems: characteristics, testing, and evaluation. Wiley:1968.

Lancaster F.W.(1969). "MEDLARS: Report on the evaluation of its operating efficiency" Am.Doc,20, pp119-42.

Lancaster F.W.(1972). Vocabulary control for information retrieval. Information Resources Press: 1972.

Lancaster F.W. & E.G.Fayen(1973). Information retrieval on-line. Melville Publishing Co.: 1973.

Lefkovitz D.(1969). File structures for on-line systems. Spartan Books: 1969.

Long P.L. & F.G.Kilgour(1972). "A truncated search key title index" JOLA,5, pp17-20.

Lord Todd(1967). "Introduction: the problem stated" in de Reuck & Knight(1967) q.v., pp4-15.

Lum V.Y. & P.S.T.Yuen & M.Dodd(1971). "Key-to-addess transform techniques: a fundamental performance study on large existing formatted files" CACM,14, pp228-39.

Lyons J.(1968). Introduction to theoretical linguistics. Cambridge Univ. Press: 1968.

Maron M.E. & J.L.Kuhns(1960). "On relevance, probabilistic indexing and information retrieval" JACM,7, pp216-44.

Martin T.H., J.Carlisle & S.Treu(1973). "The user interface for interactive bibliographic searching: an analysis of the attitudes of nineteen information scientists" JASIS,24, pp142-7.

McCarn D.B. & C.R.Stein(1967). "Intelligence systems evaluation" Electronic handling of information: testing and evaluation. A.Kent et al (Ed.). Academic Press: 1967, pp109-22.

McCarthy J., P.Abrahams, D.Edwards, T.Hart & M.Levin(1962). LISP 1.5 programmer's manual. MIT Press: 1962.

McCracken D.D. & U.Garbassi(1970). A guide to COBOL programming. 2nd Ed., Wiley: 1970.

Menzel H.(1967). "Planning the consequences of unplanned action in scientific communication" in de Reuck & Knight(1967) q.v., pp57-77.

Miller G.A.(1968). "Psychology and information" Am.Doc,19, pp286-9.

Minker J. & S.Rosenfeld (Eds) Proceedings of a symposium on information storage and retrieval. ACM: 1971.

Minker J., G.A.Wilson & E.Peltola(1973). "Document retrieval experiments using cluster analysis" JASIS,24, pp246-60

Minsky M.(1968). "Introduction" Chapter 1 in Semantic information processing. M.Minsky(Ed.), MIT Press: 1968, pp1-32.

Montgomery C.A.(1969). "Automatic language processing" Annual Review of Information Science and Technology,4, pp145-74.

Montgomery C.A.(1972). "Linguistics and information science" JASIS,23, pp195-219.

Mooers C.N.(1951). "Zatocoding applied to mechanical organization of knowledge" Am.Doc,2, pp20-32.

Morgan H.L.(1970). "Spelling correction in systems programs" CACM,13, pp90-4.

Morris R.(1968). "Scatter storage techniques" CACM,11, pp38-44.

Moyne J.A.(1969). Information retrieval and natural language. Report FSC-69-5005. IBM Federal Systems, Gaithersburg: June 19, 1969.

MTS(1973). The Michigan Terminal System. Vol.1: MTS and the Computing Center. 3rd Ed. Univ. of Michigan Computing Center, Ann Arbor, Michigan: 1973.

Murphy D.L.(1972). "Storage organization and management in TENEX" Proc. AFIPS,41, pp23-32.

Murray D.M.(1970). "A scatter storage scheme for dictionary lookups" JOLA,3, pp173-201.

Needham R.M.(1965). "Applications of the theory of clumps" Mechanical translation,8, pp113-27.

Newell A.(1961). Information Processing Language - V Manual. Prentice-Hall: 1961.

Nugent W.R.(1968). "Compression word coding techniques for information retrieval" JOLA,1, pp250-60.

Olney J.C.(1962). Building a concept network for retrieving information from large libraries. Part 1. SDC Report TM-634/001/00: Jan.1962.

Organick E.I.(1972). The Multics system: an examination of its structure. MIT Press: 1972.

Overhage C.F.J. & J.F.Reintjes(1974). "Project Intrex: a general review" ISR,10, pp157-88.

Pacak M. & A.W.Pratt(1971). "The function of semantics in automated language processing" Proc. of a symposium on information storage and retrieval. J.Minker & S.Rosenfeld(Eds), ACM: 1971, pp5-18.

Parnas D.L.(1972). "On the criteria to be used in decomposing systems in modules" CACM,15, pp1053-8.

Porter R.J., J.K.Penry & J.F.Caponio(1970). "Epilepsy Abstracts Retrieval System(EARS), a new concept for medical literature storage and retrieval" Proc. ASIS, 7, pp171-2.

Price N.H., C.Bye & B.Niblett(1974). "On-line searching of Council of Europe Conventions and Agreements: a study in bilingual document retrieval" ISR,10, pp145-54.

Quillian M.R.(1968). "Semantic Memory" (Ph.D thesis) in Semantic information processing. M.Minsky(Ed.), MIT Press: 1968, pp227-70.

Resnikoff H.L. & J.L.Dolby(1965). "The nature of affixing in written English, Part 1" Mechanical translation,8, pp84-9.

Resnikoff H.L. & J.L.Dolby(1966). "The nature of affixing in written English, Part II" Mechanical translation,9, pp23-33.

Rettemeyer J.W.(1972). "File ordering and retrieval cost" ISR,8, pp79-93.

de Reuck A. & J.Knight(Eds)(1967). Communication in Science: documentation and automation. A Ciba Foundation Volume. Churchill, London: 1967.

Rickman J. & W.E.Walden(1973). "Structures for an interactive on-line thesaurus" International Journal of Computing and Information Sciences,2, pp115-27.

van Rijsbergen C.J.(1974). "Further experiments with hierarchic document clustering in document retrieval" ISR,10, pp1-14.

van Rijsbergen C.J. & K.Sparck Jones(1973). "A test for the separation of relevant and non-relevant documents in experimental retrieval collections" J.Doc,29, pp251-7.

Robertson S.E.(1969). "The parametric description of retrieval tests. Pt.1: The basic parameters"; "Pt.2: Overall measures" J.Doc,25, pp1-27, 93-107.

Robinson M.G. & D.M.Yates(1973). "The Scrapbook information system" Information Scientist,7, pp135-43.

Rocchio J.J.,Jr.(1971). "Evaluation viewpoints in document retrieval" Chapter 4 in Salton(1971) q.v., pp68-73.

Salton G.(1962). "Manipulation of trees in information retrieval" CACM,5, pp103-14.

Salton G.(1966). "Data manipulation and programming problems in automatic information retrieval" CACM,9, pp204-10.

Salton G.(1968). Automatic information organization and retrieval. McGraw-Hill: 1968.

Salton G.(Ed.)(1971). The SMART retrieval system: experiments in automatic document processing. Prentice-Hall: 1971.

Salton G.(1972). "A new comparison between conventional indexing (Medlars) and automatic text processing (SMART)" JASIS,23, pp75-84.

Salton G.(1973). "Recent studies in automatic text analysis and document retrieval" JACM,20, pp258-78.

Salton G. & E.H.Sussenguth,Jr.(1964). "Some flexible information retrieval systems using structure matching procedures" Proc. AFIPS,25, pp587-97.

Salton G. & C.S.Yang(1973). "On the specification of term values in automatic indexing" J.Doc,29, pp351-72.

Santos C.S. & A.L.Furtado(1972). G/PL/I - Extending PL/I for graph processing. Monographs in computer science and computer applications No.11/72, Computer Science Dept., Pontificia Universidade Catolica do Rio de Janeiro: 1972.

Science Research Council(1973). Artificial intelligence: a paper symposium. SRC: April 1973. Including "Artificial intelligence: a general survey" by Sir James Lighthill.

Senko M.E., E.B.Altman, M.M.Astrahan & P.L.Fehder(1973). "Data structures and accessing in data-base systems" IBM Systems Journal,12, pp 30-93.
"I. Evolution of information systems" pp30-44.
"II. Information organization" pp45-63.
"III. Data representations and the data independent accessing model" pp64-93.

Siegel S.(1956). Nonparametric statistics for the behavioural sciences. McGraw-Hill: 1956.

Simmons R.F., J.F.Burger & R.M.Schwarcz(1968). "A computational model of verbal understanding" Proc. AFIPS, 33, pp441-56.

Simmons R.F. & J.Slocum(1972). "Generating English discourse from semantic networks" CACM,15, pp891-905.

Simon H.A.(1965). The shape of automation for men and management. Harper & Row: 1965.

Snowdon R.A.(1974). Interactive use of a computer in the preparation of structured programs. Ph.D. Thesis, University of Newcastle upon Tyne: June 1974.

Sparck Jones K.(1965). "Experiments in semantic classification" Mechanical translation,8, pp97-112.

Sparck Jones K.(1970). "Some thoughts on classification for retrieval" J.Doc,26, pp89-101.

Sparck Jones K.(1971). Automatic keyword classification for information retrieval. Butterworths: 1971.

Sparck Jones K.(1972a). "Some thesauric history" Aslib Proceedings,24, pp400-11.

Sparck Jones K.(1972b). "A statistical interpretation of term specificity and its application in retrieval" J.Doc,28, pp11-21.

Sparck Jones K.(1973a). "Collection properties influencing automatic term classification performance" ISR,9, pp499-513.

Sparck Jones K.(1973b). "Does indexing exhaustivity matter?" JASIS,24, pp313-6.

Sparck Jones K.(1973c). "Index term weighting" ISR,9, pp619-33.

Stevens M.E., V.E.Giuliano & L.B.Heilprin(Eds)(1965). Statistical association methods for mechanized documentation. Symposium proceedings, Washington 1964. National Bureau of Standards Misc. Pub. 269: 1965.

Stiles H.E.(1961). "The association factor in information retrieval" JACM,8, pp271-9.

Svenonius E.(1972). "An experiment in index term frequency" JASIS,23, pp109-21.

Szanser A.J.(1973). "Bracketing technique in elastic matching" Computer Journal,16, pp132-4.

Tagliacozzo R., M.Kochen & L.Rosenberg(1970). "Orthographic error patterns of author names in catalogue searches" JOLA,3, pp93-101.

Tague J.(1970). "Association trails" Encyclopedia of library and information science. Marcel Dekker: 1970.

Treu S.(1970). Supplementing human memory by means of interactive, computer-based associative storage and retrieval. Ph.D. Thesis, University of Pittsburgh: 1970.

Treu S.(1971). "A conceptual framework for the searcher-system interface" in Walker(1971) q.v., pp53-66.

University of Newcastle upon Tyne. Computing Laboratory (1972). PL360 programming manual.

University of Newcastle upon Tyne. Computing Laboratory (1974). Medusa information retrieval service user manual. Prepared by J.Alan Hunter.

Vernimb C.O. & G.Steven(1973). "'ENDS' - European Nuclear Documentation Service" Nuclear Engineering & Design, 25, pp325-33.

Vickery B.C.(1973). Information systems. Butterworths: 1973.

Wagner R.A.(1974). "Order-n correction for regular languages" CACM,17, pp265-8.

Walker D.E.(Ed.)(1971). Interactive bibliographic search: the user/computer interface. Proc. of a workshop on "The user interface for interactive search of bibliographic data bases", Palo Alto, Calif., Jan.1971. AFIPS Press: 1971.

Weiler G.(1973). "Relevance again" (Letter) ISR,9, p121.

Weinberg B.H.(1974). "Bibliographic coupling: a review" ISR,10, pp189-96.

Weizenbaum J.(1963). "Symmetric list processor" CACM,6, pp524-44.

Wente, van A.(1971). "NASA/RECON and user interface considerations" in Walker(1971) q.v., pp[illegible]-104.

Whitehall T.(1974). "A thesaurus for the user" Informatics 1 q.v., pp135-44.

Williams J.H.,Jr.(1969). BROWSER: an automatic indexing on-line text retrieval system. Annual Report. IBM Federal Systems Division: 1969. AD 693 143.

Williams J.H.,Jr.(1971). "Functions of a man-machine interactive information retrieval system" JASIS,22, pp311-7.

Wilson P.(1973). "Situational relevance" ISR,9, pp457-71.

Winograd T.(1972). Understanding natural language. (Ph.D. Thesis, MIT). Edin. Univ. Press: 1972.

Wirth N.(1968). "PL360, a programming language for the 360 computers" JACM,15, pp37-74.

Wirth N.(1971). "Program development by stepwise refinement" CACM,14, pp221-7.

Wirth N. & C.A.R.Hoare(1966). "A contribution to the development of ALGOL" CACM,9, pp413-32.

Woodward P.M. & S.G.Bond(1974). Algol 68-R users guide. 2nd Ed. HMSO: 1974.

Wright M.A.(1960). "Mechanizing a large index" Computer Journal,3, pp76-83.

Yngve V.(1963). COMIT programmers reference manual. MIT Press: 1963.

Zunde P.(1971). "Structural models of complex information sources" ISR,7, pp1-18.