This blank page was inserted to preserve pagination.

BOUNDS ON INFORMATION RETRIEVAL EFFICIENCY IN STATIC FILE STRUCTURES

Terry A. Welch

June 1971

PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Cambridge Massachusetts 02139

ACKNOWLEDGMENTS

I greatly appreciate the help received from Prof. Peter Elias in the course of this research. His encouragement, clear insight, and always accurate criticisms have been most valuable.

I also appreciate the encouragement and support given by Prof. J.C.R. Licklider of Project MAC.

My readers, Prof. R.N. Spann of RLE and Dr. M.M. Kessler, Associate Director of Libraries at MIT, were very helpful in forming the presentation of the material. Similar help was received from Richard Marcus and Peter Kugel of Project Intrex.

The preparation of this document has been greatly expedited by the good typing and spelling of Miss Joanne Knowlton.

Work reported herein was supported in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).

BOUNDS ON INFORMATION RETRIEVAL EFFICIENCY IN STATIC FILE STRUCTURES*

Abstract

This research addresses the problem of file organization for efficient information retrieval when each file item may be accessed through any one of a large number of identification keys. The emphasis is on library problems, namely large, low-update, directory-oriented files, but other types of files are discussed. The model used introduces the concept of an ideal directory against which all imperfect real implementations (catalogs) can be compared. The use of an ideal reference point serves to separate language interpretation problems from information organization problems, and permits concentration on the latter. The model includes a probabilistic description of file usage, developed to give precise definition to the range of user requirements. The analysis employs mathematical tools and techniques developed for information theory, such as the entropy measure and the concept of an ensemble of possible file items.

The principal analysis variable is item relevance, the probability that a file item accessed is actually useful, which is a measure of retrieval efficiency. An upper bound on average relevance is derived, and is found to give useful results in two areas. First, it shows that retrieval efficiency is determined primarily by catalog size (amount of information stored) and user question statistics, with only second-order effects due to type of catalog data and file structure used. Second, it is used to evaluate various indexing procedures proposed for libraries and to suggest improved experimental procedures in this field.

*This report reproduces a thesis of the same title submitted to the Dept. of Electrical Engineering, Massachusetts Institute of Technology, in partial fulfillment of the requirements for the degree of Doctor of Philosophy, May 1971.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 INTENT

This research arose from the conjecture that file access efficiency is not greatly affected by the methods of file organization used. A wide range of library catalog construction techniques have been proposed in the literature (such as clustering, factor analysis, stemming, automatic indexing), and they all seem to reach about the same degree of effectiveness. However, there has been no way to compare these systems and test the conjecture.

In essence, the research described here consisted of forming statistically-described data bases (files) on paper, and performing retrieval experiments using a variety of file organizations on the same data. These experiments were performed in equations rather than simulations, using the statistical tools of information theory. The use of conjectural data bases, in which correct results are known in advance, avoids the pitfalls of interpreting the semantics of user questions, and permits concentration on the questions of organizing and representing the data.

The results were as expected, showing that average relevance of retrieved file items (or equivalently, length of file search) was determined primarily by the absolute size of the file directory, measured in terms of the information it contained. The analysis also serves to indicate the second-order effects due to type of

which the user can name directly, but are rather poor at retrieving facts irrespective of their location in specific documents. The latter problem is the real challenge.

4. Static files, with a low update rate. If questions of high change rates are introduced, further problems arise which are not really related to the analysis given here.
5. Theoretical systems, rather than specific hardware or software configurations. Specific details of implementation are ignored here because of the confusion they introduce. Rather, attention is concentrated on those design problems which are common to all file systems.

1.3 THE IMPORTANCE OF INFORMATION RETRIEVAL RESEARCH

The basis of need for the research follows from three generally accepted observations:

A. Present means of access to library information are quite poor, especially for technical subject matter. The system works fairly well for document retrieval when the identity of the document is well specified, but it works rather poorly for fact retrieval in which the identity of the document is neither known nor important.

B. Future increases in published information promise to be substantial, thus overburdening an already inadequate system.

C. Computer technologies and techniques give promise of becoming economical enough to provide a means for breaking through

reasonable approach, the interpretation question looks uninteresting for further theoretical research at this time.

The storage question is quite a different matter. Here the possibilities of quantitative analysis are good, and largely unexplored. In particular, the question of file organization will become a critical one as on-line retrieval is achieved. The bottleneck in access to stored information will grow as the result of two effects: 1) Automation of libraries means the concentration of data into a single storage unit with very few access ports, as opposed to the parallel access of library stacks, and 2) the emphasis on on-line interaction increases the number of accesses to storage, as compared to batch retrieval where few accesses are possible. Thus the upcoming use of on-line computers will place a heavy load on the file access mechanism, and this will become the system parameter limiting the number of users who can be served.

The interactive nature of coming systems will also change user operating procedures, and the file organization must change to fit these new patterns. Batch retrieval systems have emphasized high recall at the expense of overloading the user with many irrelevant documents. When on-line, the user will desire very few documents at a time, particularly since his read-out device has limited bandwidth. Manual libraries provide very coarse indexing, because of cost. The interactive user will require more precise and extensive indexing information, so that he can narrow his requests down to a few documents, and computer costs will allow this. Thus, file organizations must evolve to give greater efficiency in handling

more frequent requests, smaller requests, and bigger attribute sets. This will require improved knowledge of the effectiveness of various organizations, and may require the development of new techniques.

The objectives of this research in files can be summarized in the form of three sets of questions:

1. Parameters - What are the parameters by which to measure the performance of an information filing system? What are the variables that contribute to the cost of the system? What characteristics of the document collection are important in influencing the difficulty of file organization? Past experimentation in information retrieval has been hampered at least partially because no one knows how to measure the value of a system to the user.
2. File Organization - What techniques or file organizations are useful in organizing computer files? How effective is the technique of introducing redundancy into the file? How are directories best employed? There exist many possibilities of new organizational techniques which show promise, but are difficult to evaluate without theoretical help.
3. Classification - What strategy should be used in selecting index categories to give the most user flexibility at least cost? Should categories be disjoint or overlapping; large or small? When should categories be merged or separated? The set of index terms to be used is not an inherent property of a document set, as is often assumed, but is quite open to manipulation.

These three problem areas are considered to be approximately equal in importance. They are recognized to be strongly interrelated, so the answers cannot always be simple ones.

The theoretical work to date in information filing techniques has fallen into two schools of thought, Classical and Statistical. Both schools are working in information retrieval using tools borrowed from other fields and both schools encounter trouble because the tools do not quite fit.

The classical school has been concerned with adapting standard data-processing filing methods (e.g. hash-coding) to information retrieval, without developing much of a theoretical framework for analysis of these methods. The major difficulty encountered in this approach is that most present filing algorithms depend on receiving the complete description of a requested item, while information retrieval faces the more difficult problem of user requests which are stated in terms of a small and unpredictable part of the total document description. As a result, the use of standard filing techniques produces a very expensive system, or more often an inadequate system.

The statistical school has been concerned with classification procedures using statistical tools developed in psychological testing and refined in pattern recognition. This work has been largely theoretical due to the lack of suitable data bases for experimentation. This approach has been disappointing, probably because of scale factors. That is, pattern recognition works with objects in the hundreds, while information retrieval works with hundreds of thousands.

Pattern recognition aspires to a 99% or greater accuracy, while information retrieval is happy with 50%. The statistical classification techniques concentrate on obtaining the correct class of a pattern, while retrieval requires that a document be in a reasonable class. This confusion in objectives has caused retrieval statisticians difficulty in evaluating the worth of their classification algorithms, because they are measuring their results against inappropriate goals.

The differences of emphasis which distinguish information retrieval from the related fields are really only apparent in an automated retrieval system. Previous organization and classification techniques were sufficient in manual systems because the volume of data was relatively light, and flexibility was very restricted. In large file systems with greatly expanded indexing, the unique aspects of information retrieval become magnified, and it becomes necessary to evaluate file systems under a new set of constraints. The appearance of this shift in goals exemplifies the rethinking which will be necessary as computers find their place in retrieval systems.

1.4 OUTLINE

An attempt has been made to make each of the chapters of this document self contained, so that individual sections can be read without having to understand all other sections. In particular, the mathematical derivations of Chapter Four may be skipped by the casual reader.

Chapter Two summarizes the entire work, with other chapters filling in the details. Chapter Three, showing library applications of the mathematical results derived, preceeds the actual derivations in Chapter Four because knowledge of the applications makes the mathematics more meaningful. Even non-librarians would do well to browse through Chapter Three before attacking the mathematics following.

Chapter Five enters into conjecture about the larger picture of file access, and cites some partial results as well as describing possible further research areas. An Appendix is attached which describes an alternate approach to file analysis which was somewhat less successful. However, it does provide two key results described in Chapter Five, and may be interesting to some researchers as a different conceptual tool.

CHAPTER TWO

SEARCH LENGTHS IN FILE SYSTEMS WITH DIRECTORIES

The quality of a directory (catalog) in a file system is measured by how well it narrows the average field of search in the main file for an ensemble of user questions. An ideal directory will pinpoint every relevant item in the file for every user question, while a less perfect directory will supply a longer list of items, including some irrelevant entries. A quantitative statement of expected search length for a given directory requires an estimate of two parameters:

1) Size of the ideal directory, reflecting the number of file items and the statistics of the descriptive attributes used to form questions, and
2) Statistics of expected usage, namely the relative frequency of all questions which might be asked and the expected recall level (completeness of search) desired by each user.

When these parameters can be established, it can be shown that the following bound is applicable to a range of interesting file cases:

$$-\overline{p_n \log p_n} \geq K \cdot [H(F)-H(G)] \qquad (2.1)$$

where: K is a constant depending on question and file statistics

H(F) is a measure of the ideal directory size

H(G) is a measure of the size of an imperfect actual catalog in use, measured in bits of storage.

p_n is the relevance of each item retrieved (probability that the retrieved item actually fits the question asked by the user).

This gives a lower bound on average -log p_n as a function of directory size H(G). As directory size gets smaller, the bounding value gets larger, thus requiring smaller p_n (lower average relevance) to meet the bound conditions.

When the resulting numbers are analyzed, two conclusions appear:

1) The first-order influence on average search length is simply the size of the directory; directories with low information content will give poor average search times no matter how brilliantly they are organized.
2) Variations in directory organization affect search length through a) more efficient utilization of the bits of H(G), and b) better matching the directory of the pattern of user requests, especially the level of recall specified.

A third conclusion is that past experimental technique in the information retrieval field has been inadequate. In comparing this theory with previous experimental results, it was found that the experimenters failed to understand all the parameters they were faced with, and did not hold enough parameters constant to obtain meaningful results.

The following pages elaborate on the above theme, describing the file model used, deriving some mathematical relationships, and applying the results to practical library organizations.

important feature of eliminating semantic interpretation as a variable in the analysis. It is assumed here that the user already understands the attribute meanings, and that the relation between attributes and items has already been correctly determined. This permits concentration on the organizational aspects of filing.

Implementation (Real Directory). Actual library catalogs contain much less information than an ideal directory. Typically, they will omit information for some attributes, or will store combinations of attributes where one description is used to represent all attributes in a group (forced synonyms). The important parameter of a catalog is how many bits are used to store it; the means of best using these bits are discussed later.

Directory Output. The directory supplies a search list of all items which possibly fit the user question. Each item in the list has a probability, p_n, of matching the question. For the ideal directory, $p_n = 1$ or 0, since accurate retrieval is assumed. The sum of p_n over all items in the list gives a constant, because the number of items in the file which fit the question is fixed regardless of what directory implementation is used. Good directories concentrate the sum of p_n in a few items, while poor directories spread it out over a longer search list. The items in the search list are ordered according to decreasing p_n, so that the user can search through the most likely items first and then progress through less likely items until his recall level is reached.

Ensemble of Files. To measure the complexity of the ideal directory, a key assumption must be made. It is that we are dealing with an ensemble of files, not just one configuration. The ensemble is the set of possible file configurations which the user might expect to appear in a given library. While only one such configuration actually does occur there, the user does not know which one that is until after that information is conveyed to him.

Each possible file has an ideal directory. Not all directories are equally likely to occur, but the distribution of occurrence probabilities can be predicted from knowledge of attribute statistics and inter-attribute correlations. Information theory provides a handy measure of the variety of possible combinations, the entropy function, which happens to measure the minimum number of bits required on average to store the directory.

Thus for a file directory F, given a knowledge of the expected frequency of occurrence of the various configurations of F, the entropy H(F) can be calculated. Similarly, H(G) measures the configurations which an implementation G can take on, weighted by their frequency of occurrence. The exact value of H(F) for a real file is difficult to calculate or even guess at. Fortunately, the value of H(F) used need not be very precise, and in many cases it is not needed at all as long as H(F) is larger than H(G). H(G) is more easily measured, being the number of bits used to store the catalog (a design parameter).

2.2 MATHEMATICAL ANALYSIS

For a limited number of file and question ensembles, a rigorous statement can be made about the average of log p_n.

$$- \overline{p_n \log p_n} \geq K[H(F)-H(G)]$$

where K depends on file and question statistics; and equality occurs under the conditions that:

a) each item in F has a unique representation in G, meaning there is no ambiguity in the indexing process which creates the directory.

b) the members of G can be described without correlations, which means that the information in G is not wasted describing inter-attribute mutual information.

Implications. This relation says that a given directory size, H(G), determines a lower bound on -log p_n averaged over all questions and item configurations. As the bounding value goes up, the values of p_n (relevance) must get smaller to preserve the inequality. The average value of p_n is thus set by H(G) alone, and file organization can only serve to adjust the distribution of p_n values amongst the items.

For user questions desiring total-recall, the best distribution is to have all non-zero p_n be equal, which minimizes the number of non-zero p_n's (items which must be searched to achieve total recall). The bound (2.1) can then be solved to show, for simple cases:

of the conditions causing inequality. None of these results would surprise an experienced librarian.

File Ordering. An interesting result of this model and approach is the conclusion that the ordering of items in the file (shelf sequence) can be analyzed as part of the directory function. The information used to determine file ordering is indistinguishable from information stored in the directory, so H(G) is actually the sum of these two informations. Thus all the trade-offs in determining p_n's apply to the selection of criteria for file ordering as well as to selecting attribute information for the directory.

Exceptions. The above conclusions are based on a relation derived for a limited set of file cases:

1) The derivation assumes that attributes are statistically independent in F and Q, and that each attribute is described by the same set of usage statistics. If correlations or unequal usage occur, then the bound value is calculated with modified entropy functions: H(F')-H(G'). In H(G'), information is weighted differently for each attribute; information put into heavily weighted attributes is more effective than that put into attributes of lesser weight.

2) The derivation assumes a particular type of implementation strategy to minimize search times. This strategy seeks to minimize attribute correlations in the description

of G, which has the effect of minimizing the number of attribute ambiguities in G (this is better described in Chapter Four). I hypothesize that this strategy is optimal over all F's and Q's, but this has not been proven. No examples have been found which contradict this hypothesis, but if they were found, the relation 2.1 would not apply as an inequality but only as an estimate. Any error in such a case would be second order for normal file statistics, and the general conclusions drawn above remain intact (although some of the cataloging criteria of Chapter Three might be invalid).

2.3 IMPLICATIONS FOR LIBRARIANS

The above thinking processes put a somewhat different light on many of the problems which occupy researchers in information retrieval. The problem of how to build the best possible catalog actually becomes a problem in trying to figure out what the user really wants. This is not easy in practice because users have been so well trained by inadequate present systems that they simply do not think to go to the library with most of their questions, and so these questions cannot be counted.

As a sample of how library problems can now be viewed, consider three issues of interest:

1. Attribute Selection. Should attributes be broadly defined short terms (keywords) or more precise terms of less frequent usage (subject phrases)? Given that the same amount of catalog information is gathered under each system, the difference is only that:

   Short terms give a little information for a lot of items, minimizing the length of high-recall searches but not giving any preference to low recall searches. High precision terms serve a narrower field well, giving very good relevance in a few cases (benefiting low recall questions) at the expense of little help for total-recall questions.

   No particular attribute precision level is the right one for all systems, for user recall patterns must decide this issue.

2. Clustering. Clustering usually takes the form of grouping ideal attributes into a set which can be represented by a single composite attribute (typically the union of the set). This is a useful form of information reduction which brings the amount of data down to the level that can be stored in the catalog. It serves to create lower-precision composite attributes so that high-recall searches are aided, at the expense of low-recall searches. It serves to combine correlated attributes so that directory entropy is used most efficiently. These virtues, however, occur to a limited extent, and existing clustering algorithms probably

approach the theoretical limit of clustering effectiveness.

3. Thesaurus. A Thesaurus is an additional means of putting information into a catalog, and one which is a convenient form of coding information from semantically similar attributes. It is useful to A) extend the range of attributes which were only partially indexed over the document set, and B) to give entry to attributes otherwise not appearing in the catalog. As an optional tool to use when reaching for high recall, it can only improve relevance. When building a system, however, the thesaurus must be compared against the alternate possibility where the same time and money is put into just adding more index information.

Experimental Procedures. The experimental work performed to date in information retrieval research has not lived up to expectations, partially due to careless experimental technique (detailed in Chapter Three). The parameters of the experiments have not been adequately defined, with the result that too many parameters are varied at once, and nothing is learned. The principal fault observed was where two indexing languages were compared in trials where the two directory sizes were unequal, and the language blessed with the larger directory is always found to be "the best". A second error mode is to compare two languages which give different relevance distributions (one favoring high-recall questions and the

other favoring low recall), and conclude that one is superior when in truth they are equal but adjusted to different user question ensembles.

Library Strategy. The obvious conclusion of the bound derived here (2.1) is that libraries can be improved only by great additions of index information to catalogs. This approach, however, has its drawbacks. First is the problem of search time through the expanded catalog. Second is the cost of indexing, which is probably proportional to H(G), and is prohibitively high. The possible solution to both these problems lies in computers, through fast search of machine-language catalogs, and through automatic indexing. It may be a while before this is an economical solution, however.

The advent of computers in retrieval will shift the emphasis in file organization. As interactive computer systems are better utilized to overcome the semantic problems of user question interpretation, the demands on the file will shift to lower-recall question distributions, requiring a different approach to attribute selection and utilization in the catalog.

CHAPTER THREE

APPLICATION: LIBRARY ORGANIZATION

The problem of the library is that it is effective for document retrieval but not fact retrieval. Most of the subject topics in the general literature do not fall into simple subject categories, are not confined to well-indexed reference books or collections, and are not described by a few source documents. For example, the topic "theory of file organization" is related not only to libraries, computers, and information theory, but also linguistics, matrix theory, and chemical abstracts. The user who needs information about such dispersed topics faces virtual exhaustive search of the library, so he takes his questions elsewhere or leaves them unasked.

How do we provide users access to all the information in a library? How can a user find a fact in a document whose title does not describe or even imply the existence of that fact? What form of catalog or classification mechanism is necessary to provide flexibility in user specification of his information needs?

Numerous workers in the field have suggested file organization or information classification procedures which purport to attack this problem. Virtually all these systems are untested, because there exists no convenient means to compare them or predict their usefulness in a real file. The resulting confusion is increased by the failure of the researchers to define the exact problem they are solving, and indeed no two of them seem to be working on quite the same problem. A prerequisite, then, for analysis of these various

organization methods is a careful understanding of the problem and its parameters.

To this end, this chapter builds a model of a file and its access procedures, with emphasis on providing the user extensive flexibility in specifying questions. The subsequent mathematical analysis describes the trade-offs available in file organization. The results are used not only to compare the theoretical value of the various information organization schemes but also to criticize experimental work in the field.

3.1 A DIRECTORY MODEL

The following pages describe a model, which hopefully is more than a mere exercise in formalism. I claim that the following model is not only an adequate one, but that it precisely identifies the critical parameters of the file system. While some of the simplifications may seem arbitrary, they were necessarily chosen to achieve the best problem understanding with the least analytic complexity.

The detailed discussions which justify specific decisions made in the model have been segregated to the end of the chapter, in Section 3.9. The reader is encouraged to refer to this material if some aspect of the model is unsatisfactory to him. If the decisions made do not seem unusual, then the page references given can be ignored.

We view the library system as being a set of documents serviced by a catalog (called a directory). Each item has an identification

number (generally describing its physical location in the file), and the directory serves to suggest those items (identification numbers) which may satisfy a user question. Figure 3-1 illustrates this model. The user specifies a request, receives a list of suggested item numbers from the directory, and searches through the suggested items until his desired facts are found. The size and composition of the directory are the parameters of interest in the following discussion.

User Questions (Input to the Directory). To calculate search times in the file, we must specify exactly what range of questions users will ask a particular library. Failure to make this specification is a common weakness in file organization research.

A. Intersection Form - This model will be limited to questions formed by the logical intersection of attribute values. That is, a file item is relevant to a question only if it matches all attribute values specified. This form of question seems to be representative of most question types, and it is more easily analyzed than the rest (see page 87 for further discussion).

B. Number of Attributes - The user will select from a range of r available attributes which describe the document set. This set of r attributes is assumed to be large enough to allow any user to accurately specify his question. The size of r determines the user flexibility in accessing the file, for as r increases so does the number of different questions

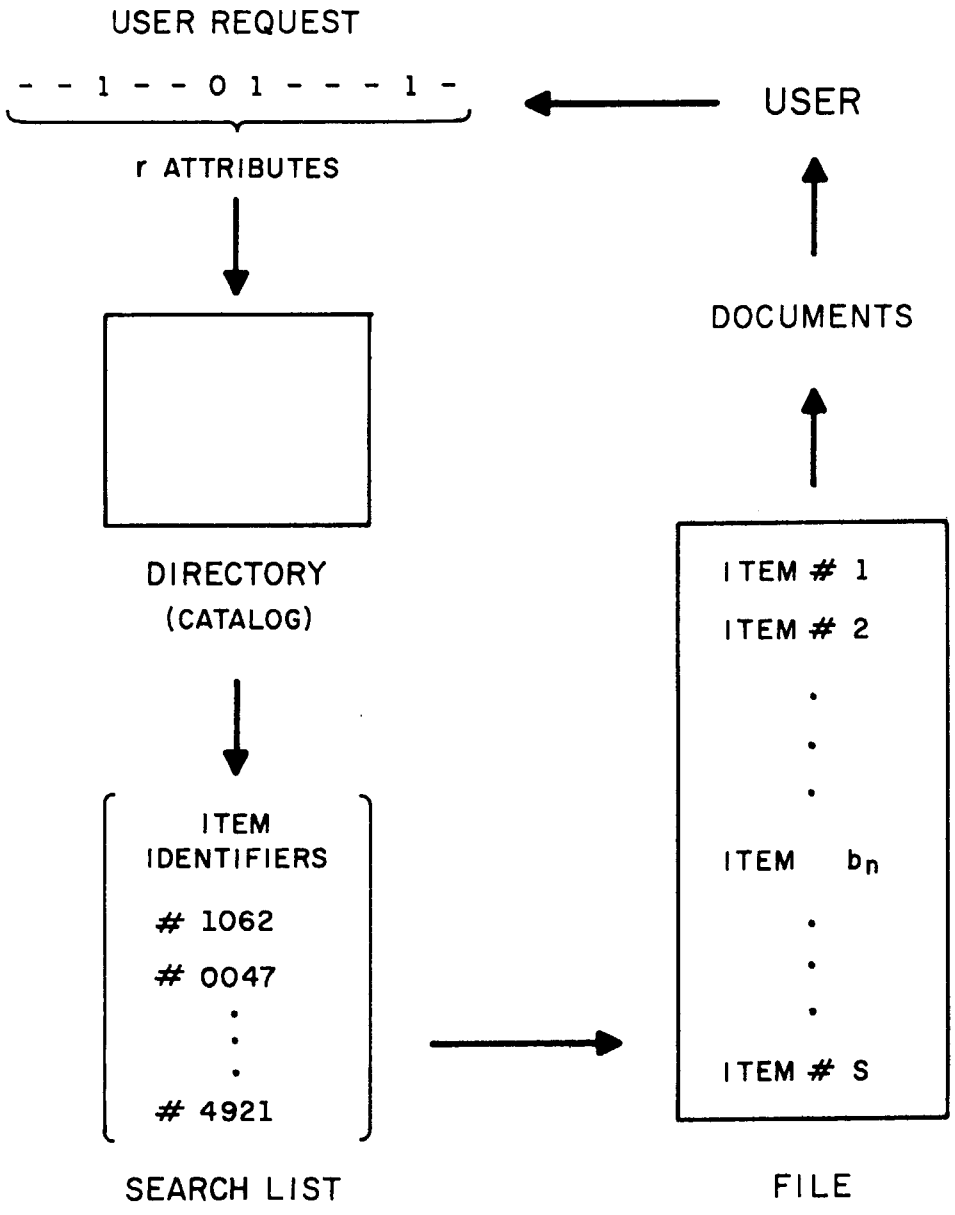

FIG. 3-1 FILE MODEL

specifies attributes independently, where for each attribute a_i there are values $P(a_i=1)$ and $P(a_i=0)$, such that $P(a_i=1)+P(a_i=0)+P(a_i=\emptyset)=1$. The interesting point about such assumptions of user statistics is that they suppose that all arbitrary combinations of attributes are used. This conflicts with the tendency in most library systems to view attributes in a hierarchical structure.

F. Recall Level - Not all users wish to see all documents which relate to their questions. Indeed a large class of inquiries seeks only one document out of many acceptable possibilities. Each question, then, is assumed to specify a recall level, indicating how many or what percentage of the relevant items are desired. Some distribution of recall levels will be assumed, for example saying that each recall level is equally likely, or that low recall questions occur more frequently than high recall questions. This parameter is important because a low average recall makes file organization much easier.

Search List (Directory Output). The response to a user question is a list of all items which are possibly relevant to the request. The user searches through these items in the file until his recall level is reached. Several parameters can be quantified in this process:

A. Relevance Probability (Efficiency) - Each item, b_n, in the list has a probability p_n, $0 \leq p_n \leq 1$, of actually being relevant

to the question. For example, a poor directory will produce a long list of items with low p_n's, while a good directory will produce a short list with higher p_n's. Most present catalogs do not supply this probability explicitly for the user, but he is often able to make rough calculations based on the attribute fit. The value p_n is a direct measure of the efficiency of retrieving an item from the file.

B. Item Sequence - Items in the search list are ordered by decreasing p_n, so that the most likely items are examined first.

C. Relevance Curve - If the various p_n's are plotted for the items in the search list, the resulting curve has two properties:

   a: It is monotone decreasing, by definition of the sequence ordering,

   b: The area under the curve equals the expected number of file items relevant to the question asked. For a given question and a given file (document collection) the area of the curve will be constant regardless of the directory used.

This relevance curve is a principal analytic tool in the following discussion. It is not quite the same as, but very similar to, Salton's recall-precision graph.

D. Search Length - The number of items in the search list with non-zero p_n is the full-recall search length, namely the number of items which must be searched by the user. This is the measure of the cost of accessing the main file. Average search lengths can also be calculated for partial recall question sets.

Directory Contents. The directory is modelled at two levels: the ideal directory, and the actual implementation.

A. Ideal Directory - If we had a perfect directory, it would contain the precise relationship between each of the s items in the file and the r attributes of the user questions. The search list produced would always have p_n's equal to exactly 1 or 0, with no uncertainty about relevance.

It is necessary to presume such a perfect directory as an analytic tool to differentiate filing errors from the semantic errors of attribute interpretation. These two sources of errors occur, and can be corrected, independently; but they produce the same system performance degradations, so some artificial separation is necessary to reduce the complexity of the analysis (see page 89).

B. Implementation - In a real library, the catalog used is a subset of the perfect directory, containing less information. Consequently it gives uncertain answers where asked to relate some items to some questions. Typically the real catalog has less information than the perfect directory because:

1. Some attributes have been omitted,
2. Some attributes have been substituted for others not exactly their equal,
3. Relationship between certain attributes and items have been added or dropped to make the attributes easier to store,
4. Some attributes have been clustered together and replaced by a composite attribute.

Further illustrations of real catalogs occur in Section 3.3.

C. Item Independence - In the perfect directory, it is assumed that the r attribute values of a given item are not correlated with the values of other items in the file, given that the statistical properties of the attributes are already known. That is, the existence of one item description in the file does not raise or lower the probability of any other item description being in the file. Equivalently, any new item added to the file cannot be better predicted by knowing the specific existing membership of the file than by knowing the statistics of the attribute occurrences of the file in general.

This is not completely true in real situations, for it can sometimes be said that a library having item X will almost always have item Y as well as, for example, when X and Y are two volumes in a set. However, the extent of inter-item correlation is believed to be quite small, so the time required to perform a more exact analysis

would not be warranted. Further, it seems likely that the existence of inter-item correlation would not alter the general results derived herein (see page 87).

D. Information in the Ideal Directory - We assume that the person using a file knows the statistics of attribute occurrence in the subject area, but does not know which particular items appear in the file. Then the ideal directory contains the information saying which items actually do appear, and this can be calculated as the sum of the informations needed to specify each item (since the items are assumed to appear independently). That is:

$$H(F) = \sum_{n=1}^{s} H(C_n)$$

and

$$-H(C_n) = \sum_{m=1}^{2^r} p(c_m) \log p(c_m)$$

where $H(F)$ = entropy in the file F

$H(C_n)$ = entropy of the configurations of item b_n.

$p(c_m)$ = probability of an item taking an attribute configuration c_m, where c_m is one of the 2^r possible configurations in C.

This entropy can be calculated on an attribute basis as well:

$$H(F) \leq \sum_{i=1}^{r} H(a_i)$$

where $-H(a_i) = p(a_i=1)\log p(a_i=1) + p(a_i=0)\log p(a_i=0)$.

The equality holds for H(F) if the attributes happen to be statistically independent in the file.

This entropy, H(F), is the minimum number of bits needed, on average, to store the ideal directory. It gives an accurate measure of the difficulty of accessing the file, as will be demonstrated below.

E. Information in the Implementation - The actual catalog used will, in general, be much smaller than the ideal directory. This implementation is called G, and has entropy:

$H(G) \leq$ number of bits in catalog,

with equality when the catalog bits are efficiently used, namely equally likely one or zero, and independently chosen.

It is the measure of cost of building the catalog (see page 90).

In general, H(G) will be broken down on a per-item basis, similar to that of H(F):

$$H(G) = \sum_{n=1}^{s} H(E_n)$$

and

$$-H(E_n) = \sum_{k} p(e_k)\log p(e_k)$$

where $H(G)$ = entropy in implementation G.

$H(E_n)$ = entropy in the configurations representing b_n.

$p(e_k)$ = probability that an item is represented by implementation configuration e_k.

In summary, the most important aspect of this model is the definition of an ideal directory. This serves to bypass semantic interpretation problems which also contribute to retrieval errors. However, no ideal directory of a real file is available for experimentation. Therefore, we experiment with a range of hypothesized ideal directories, to study the effects of different implementations.

3.2 A BOUND ON DIRECTORY EFFICIENCY

For the above model, the following result has been proven true for all questions and all files (see Chapter 4). For any item:

$$\overline{-p_n \log p_n} = I(Q\#;C|E)$$

averaged over all questions q in Q and item configurations c in C,

where # is the event that, for each q, the item b_n fits the question.

This relation takes a great deal of explanation. The left-hand side gives a restriction on relevance curves which affects the shape of these curves, and it is a measure of the retrieval error in the system. The right-hand side says this: If an item b_n is known to fit a question q, how much more does this tell me about b_n than I already knew from the directory?

While this relation is precise and general, it is not directly meaningful and must be further interpreted for special cases. For the case where:

1) All attributes are described by the same statistics, both in the file and in user questions (i.e. $p_Q(a_i=\emptyset)$ is constant over i, etc.),

2) Each item configuration has a unique representation in E, $H(E|C) = 0$ (no ambiguity in the indexing process), and

3a) No correlations exist in E (a desirable system design goal, described later), or

3b) Questions are specified with attributes taking on values 1 or 0 equally likely, and independently,

the relation can be rewritten (details of derivation omitted here):

$$\overline{-p_n \log p_n} = K(Q,F)[H(C)-H(E)]$$

where K(Q,F) is a constant determined by question statistics and file statistics, but not implementation statistics;

K(Q,F) is determined by:

$$\overline{-p(\#)\log p(\#)} = K(Q,F)\cdot H(C).$$

File cases which do not fit the above restrictions can be viewed as variations on this basic case:

1) If some attributes are accessed more frequently than others, or have different probabilities of taking 1 or 0

values in Q or C, then the various attributes will be weighted differently in the computation of H(E). That is, information put into the directory should be put in the most-used attributes first, etc. The H(C)-H(E) term is still valid in form, but information for some attributes is weighted more heavily than others.

2) If items are not uniquely represented, then H(E) is inefficiently used, and the relation becomes an inequality giving a lower bound on $\overline{-p_n \log p_n}$.

3) If E can have correlations, and Q is not symmetrical in 1 and 0, then a more complicated situation exists. Generally, both of the previous conditions occur, namely attributes become unevenly weighted and the relation becomes an inequality, lower bounding $-p_n \log p_n$. The inequality has not been proven rigorously, but appears true for all useful file situations (Chapter Four discusses this). Whether or not precisely true for all file situations, the inequality makes an excellent working tool, to be used throughout this chapter:

$$\overline{-p_n \log p_n} \geq K(Q,F)[H(C) - H(E)]$$

From this bound, several observations can be drawn. The H(C) term is given by the user demands on the system, being defined by the set of attributes which the user would like to use. The H(E) term measures the effectiveness of the catalog in meeting the user needs, and should be made as large as possible as an objective in designing the catalog. Two lines of analysis are provided by the bound:

1. The first-order measure of catalog effectiveness is purely in the relative size of H(C)-H(E), namely in the adequacy of the catalog in terms of its absolute size. All catalogs of the same size, and of reasonable construction, will produce about the same effectiveness in file access efficiency.
2. The details of creating a catalog are concerned with approaching the lower limit of the bound as closely as possible, consistent with fitting the needs of the user. This is discussed at length below.

To give more concrete details of what the bound demonstrates, a specific example will be examined at length.

3.3 EXAMPLE OF A LIBRARY

Presume a file system having:

Documents: one thousand $s = 10^3$

Attributes: one thousand $r = 10^3$

Attributes per document: 250 $\rho = \frac{1}{4}$

Questions: All attribute combinations such that $p(\#|q) = .016$.

That is, all questions used have an expected value of 16 items which will be retrieved. Each question will have 3 attributes specified to have value 1, with each of the r attributes appearing equally often:

$$p(a_i=1) = 3 \times 10^{-3}$$

$$p(a_i=0) = 0$$

$$p(a_i=\emptyset) = 1 - 3 \times 10^{-3}$$

Explanation. This example was selected to approximate a real library situation, but with numbers chosen so that the resulting output curves are clear and meaningful. A real library would probably have more items, more attributes, and a smaller ρ.

Assume that this is just a set of books, with a very extensive subject classification system. The attributes might also be author's names, journal names, etc. The books are assumed to have attribute configurations chosen randomly and independently from the 2^r configurations permitted.

The attributes of this example will be assumed to be uncorrelated. That is, all item configurations having exactly X 1's will have the same probability of occurrence, regardless of which X attributes are involved. I apologize for this unrealistic assumption, but the example can be made more illuminating under these conditions. Variations on this example, showing more realistic conditions, are discussed in Section 3.4.

The number of questions allowed is quite large, saying that each book can, on average, be distinguished from all others by about $\binom{250}{3}$ different questions, or about 10^6 ways. This is necessary for successful access for the user.

The specification that every question should have the same $p(\#|q)$ is a convenient trick to provide clear results. If a range of question precisions were permitted, the questions with larger $p(\#|q)$ would have a disproportionate influence on the search length averages, thus obscuring the results of the example. The results obtained for constant $p(\#|q)$ questions give useful insight for other question types as well.

Entropy of the Ideal Directory. If the r attributes are mutually independent as assumed, H(F) can be calculated as the sum of the entropies of every bit relating an attribute to an item. If correlations existed, H(F) would be smaller.

$$H(F) \leq r \cdot s \cdot H(\rho)$$

where $H(\rho) = -\rho\log\rho - (1-\rho)\log(1-\rho)$.

$$H(F) \leq 10^3 \cdot 10^3 \cdot .811 = 8 \cdot 10^5 \text{ bits}$$

The exact value of H(F) is not critical, as will be observed later, unless it is close to H(G) (rarely the case).

Implementation Entropy. The entropy of the catalog is simply the number of bits we are willing to prepare and store in constructing the file system. In library catalogs this information may be stored in English, not binary, so a little interpretation will be required.

It is a reasonable working approximation to say that a book can be identified uniquely within a library through about four fields of identifiers: title, publisher/date, shelf classification/author, and subject terms. If anything, this is being a little generous to present catalogs. Using the numbers of the example, the 10^3 books in the file require that 10 bits = $\log_2 10^3$ of information be supplied to uniquely distinguish one book. Each book that can be distinguished on four fields thus has 40 bits of descriptive information in the catalog. That number does not make an interesting example, so I increase it by x5 to give:

$$H(G) = s \cdot 200 = 2 \times 10^5 \text{ bits.}$$

$$H(F) = 4 \cdot H(G).$$

This is the capacity of the catalog, not necessarily the useful information content. Poor use of the catalog information can always decrease retrieval efficiency.

Error Bound. Having determined a file size and a catalog size, we can compute a retrieval error bound:

$$\overline{-p_n \log p_n} \geq K(Q,F)[H(C)-H(E)]$$

for $p(\#) = .016$

and $H(C) = 811$ bits

$H(E) = 200$ bits

then $K(Q,F) = 12 \times 10^{-5}$

$$\overline{-p_n \log p_n} \geq 12 \times 10^{-5}[811-200] = .072$$

There are a large number of solutions which achieve equality in this bound. Some of the extreme cases are shown here. The solutions are shown as relevance curves, which give the distribution of p_n's over a full file.

Total-Recall Bound. Figure 3-2 shows a relevance curve which satisfies the $p_n \log p_n$ bound, and which gives the shortest possible average search length for 100% recall questions. It says that 355 items must be searched for each question, and there is no possible

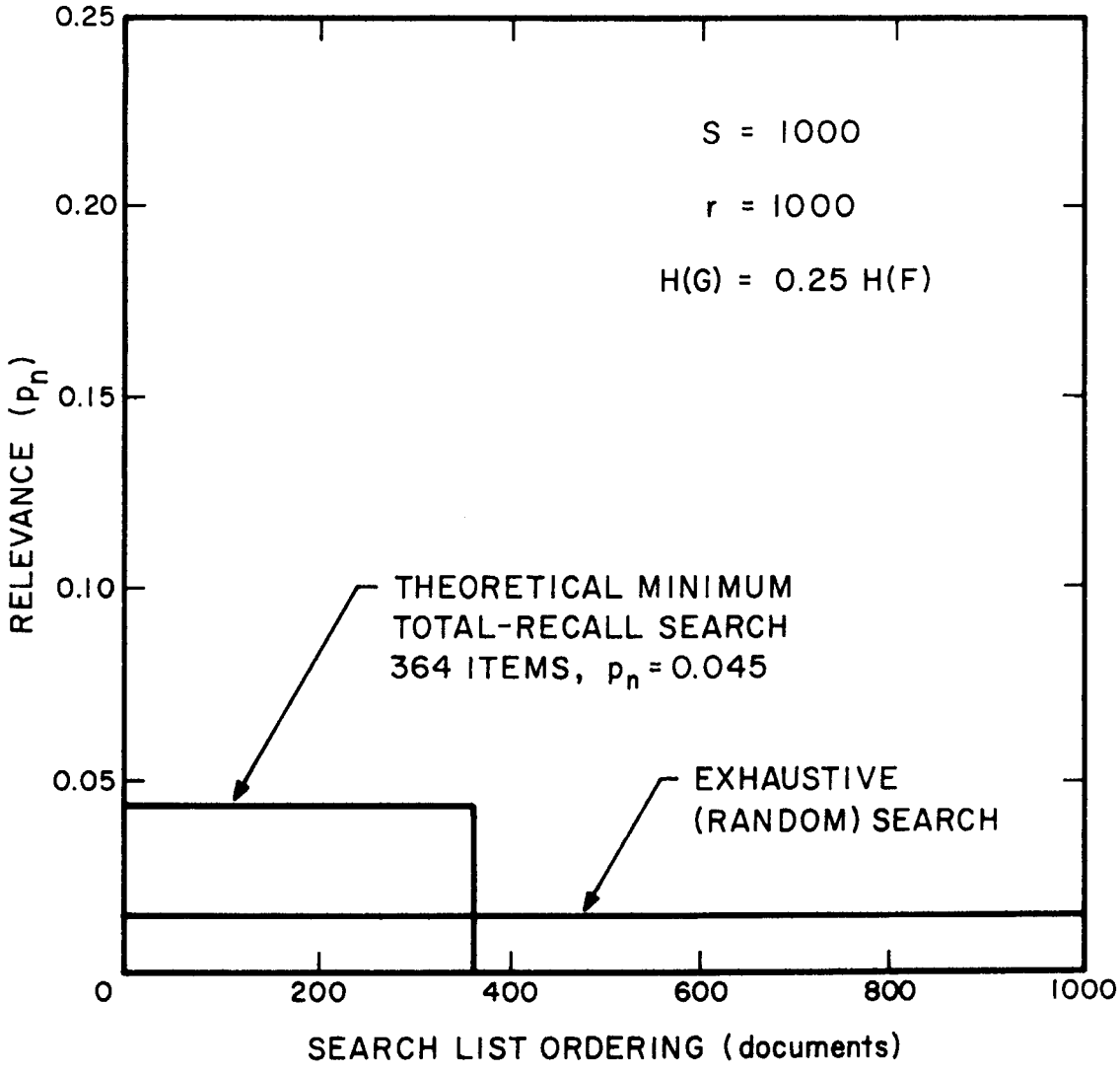

FIG. 3-2 TOTAL RECALL BOUND FOR EXAMPLE 3.3

catalog of the size specified which can give shorter average total-recall searches.

The logic behind that statement is this:

355 items at $p_n = .045$ gives:

a) $-p_n \log p_n = .201$, and $.201 \times .355 = .071$

b) expected number of relevant items: $355 \times .045 = 16$

No $p_n > .045$ can fit these two conditions. If, for example, someone proposed a catalog giving $p_n = .1$ for 160 items, $-p_n \log p_n = .033$ being smaller than the $-p_n \log p_n$ bound guarantees that no such catalog could be built.

The curve of Figure 3-2 cannot actually be achieved by any real implementation for this example, although a reasonable approximation is shown later.

Limited Attributes. One sample implementation consists of a catalog consisting of 25% of the attributes of the ideal catalog. When considering all questions asked of the file, this gives a family of relevance curves, shown in Figure 3-3. That is, if all three attributes of the question fall within the 25% of the attributes retained, then perfect access to the main file is possible. At the other extreme, if all requested attributes fall into the ignored 75% exhaustive search of the file is necessary.

This form of implementation is used to do a good job of access for a few users at the cost of ignoring the majority. Those users who can formulate their questions in terms of the 25% catalog attributes will do so, and other users will go elsewhere.

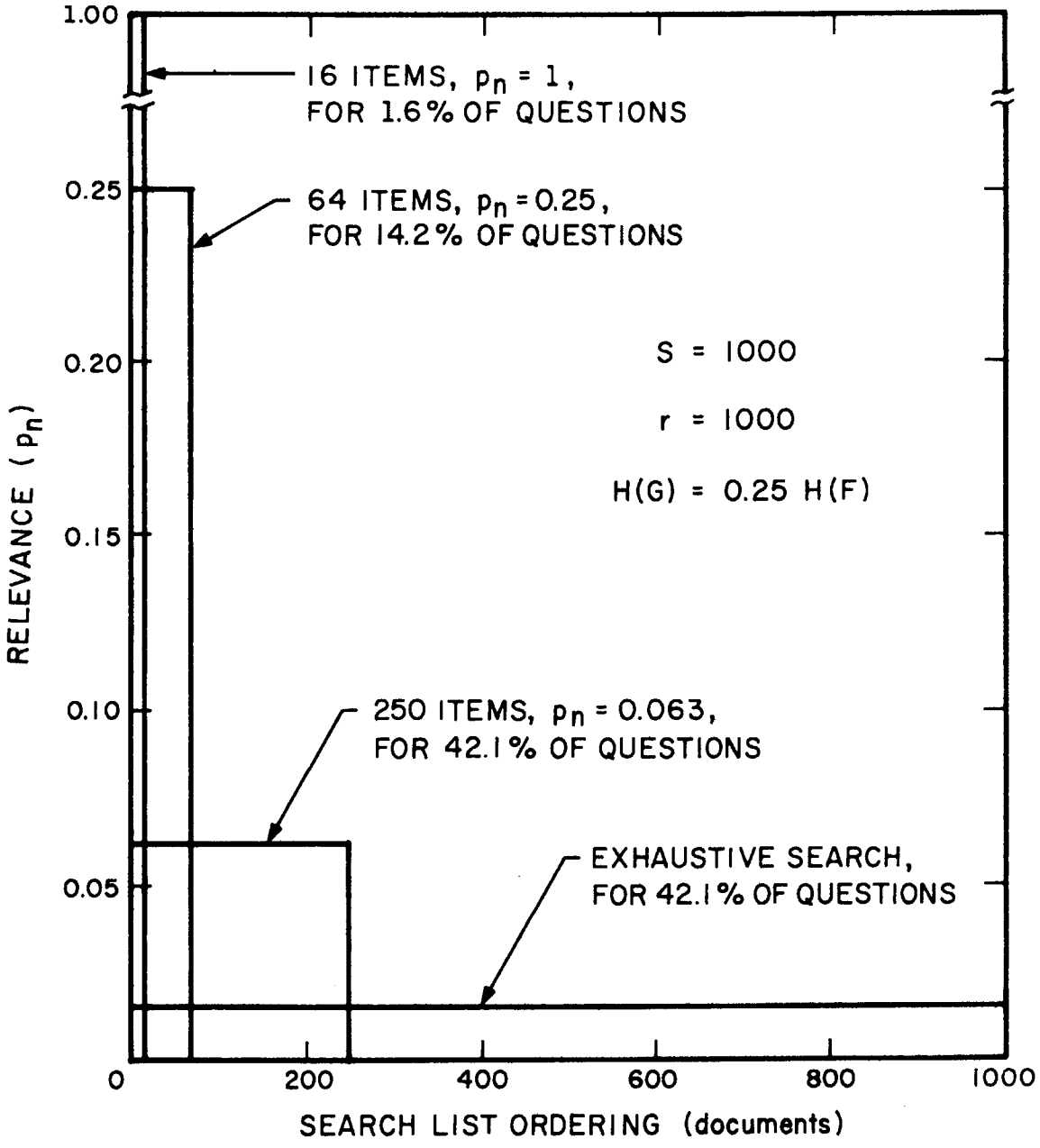

FIG. 3-3 RELEVANCE CURVES
LIMITED ATTRIBUTE IMPLEMENTATION
FOR EXAMPLE 3.3

Checking the $\overline{-p_n \log p_n}$ numbers:

p_n	$-p_n \log p_n$	%	number of items	$\overline{-p_n \log p_n}$
1	0	.016	16	0
.25	.5	.142	64	.0045
.064	.254	.421	250	.027
.016	.095	.421	1000	.040
				.0715

$$\overline{-p_n \log p_n} = .0715$$

Randomly Selected Attributes. Another catalog approach is to list 32 attributes for each book, randomly selected from the 250 attributes in each case. Thus all attributes would appear in the catalog but would be related to only 12% of their possible documents. This produces the relevance curve of Figure 3-4. This density $\rho = 032$ requires 25% of the bit storage that $\rho = .250$ attributes require.

This curve is well suited for questioners who seek only partial recall on their questions, but seldom seek full recall (because that involves exhaustive search). Random attribute selecting is quite easy in practice, for that is what happens when authors classify their own documents using a free vocabulary.

Note, however, that random selection of attributes violates one of the conditions for approaching the bound in the p log p inequality, namely $H(E|C) = 0$. With random selection, there are many configurations (e's) which legally represent any one item, with a resulting loss of information. Indeed, for this example

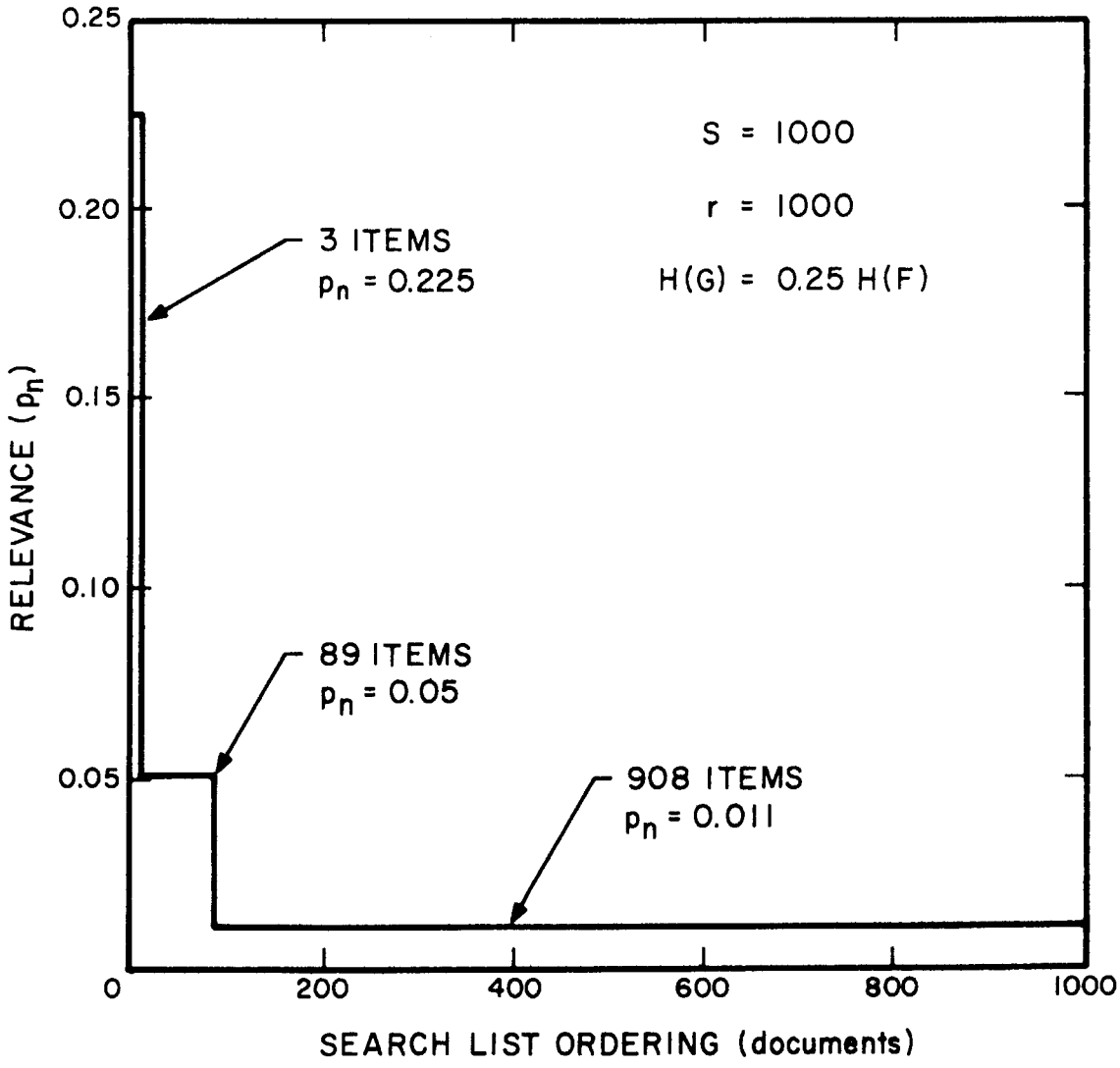

FIG. 3-4 RELEVANCE CURVE
RANDOMLY SELECTED ATTRIBUTES
FOR EXAMPLE 3.3

$\overline{-p_n \log p_n}$ = .085, indicating that about half of the directory information was inefficiently used.

To achieve a smooth distribution of relevance curves without such loss of information requires a pseudo-random attribute selection system, where catalog information is taken from all attributes but in a predictable pattern. There is little chance of this happening in a real library.

Consolidated Attributes. Figure 3-5 shows the results of combining attributes into representative groups. That is, each attribute in a group is represented in the catalog by the union (logical "or") of all attributes in that group. This effect is similar to classification methods such as "clustering" and "stemming", which are just different methods for grouping attributes. For this example, attributes were grouped randomly, since the discussion of selective grouping is delayed until later in the chapter.

In this example each composite attribute represents 4 attributes in union. The composite attribute has value zero the 32% of the time that all 4 attributes it represents are zero, and it is one otherwise. There are 250 such composite attributes, each with entropy .90 = H(.32), the total entropy of the system is 225 bits per item, 10% more than specified. Each attribute specified by the user thus eliminates 32% of the file items, and only the rest need be searched. 99% of the time the user's three specified attributes fall into separate composites, so that only $(.68)^3$ = 33% of the file is searched. 1% of the time, two of his attributes fall

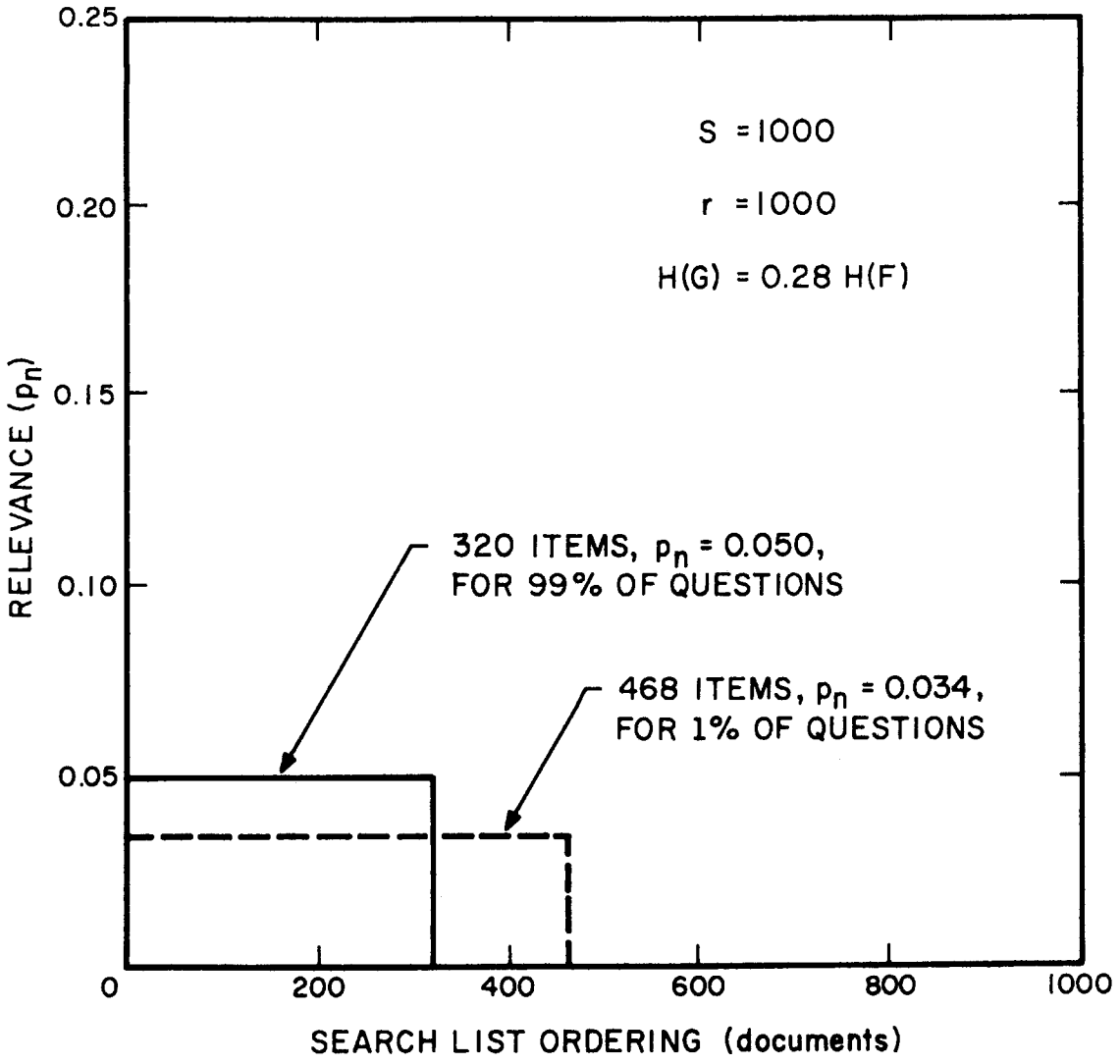

FIG. 3-5 RELEVANCE CURVES CONSOLIDATED ATTRIBUTES FOR EXAMPLE 3.3

into the same composite group, narrowing his search to $(.68)^2 = 47\%$ of the file. With probability 10^{-5}, he will have to search 68% of the file when his three attributes fall into one group.

Checking these results against the $p_n \log p_n$ restriction, we calculate:

$$-\overline{p_n \log p_n} = 70.0.$$

This is about as predicted by the bound, adjusting for the 10% larger directory.

Small Library. Another approach is to ignore 75% of the library items when constructing the directory. This gives perfect relevance for 25% of the books, and exhaustive search on the other 75%. This is possibly a reasonable approach for partial-recall questions, but is poor for total recall questions. It is what happens when a small library is formed in a specific subject area, with a knowledgeable librarian (the ideal catalog) guiding all inquiries.

Classification. Another approach to cataloging is to use attributes which are disjoint. That is, within a group of attributes each item belongs to one attribute or another but never two at a time. Such disjoint sets are created by overlooking overlaps in documents and picking one attribute only from each set. (The Dewey Decimal System is an example of one disjoint set of attributes.)

To achieve this format with the numbers of this example, the attributes have to be grouped in sets of 22:

45 sets, 22 attributes per set, each attribute is related to 45 items (18% of its true values). Retrieval for virtually all questions yields:

For		$-p_n \log p_n$	$\overline{-p_n \log p_n}$
6 items	.215	.477	2.9
129 items	.046	.204	26.3
5 items	0	0	0
all other items	.010	.066	56.8
			86

This relevance curve is shown in Fig. 3-6.

This p log p value comes out larger than the bound value because the approach violates the $H(E|C) = 0$ condition of the bound. When an item could fit under several attributes of a single classification but may be listed under only one, there is ambiguity as to which one is to be selected. As a result, the directory information is only about half utilized, giving the resulting poor retrieval.

Example Summary. The clear implications of this example are:

1) With only 25% of the possible information in the catalog, typical search lengths involve one-third of the file.
2) As a second-order effect, catalog information coding can be chosen to adapt the file to user recall needs, reducing search lengths for partial recall at the expense of lengthening searches for total recall.

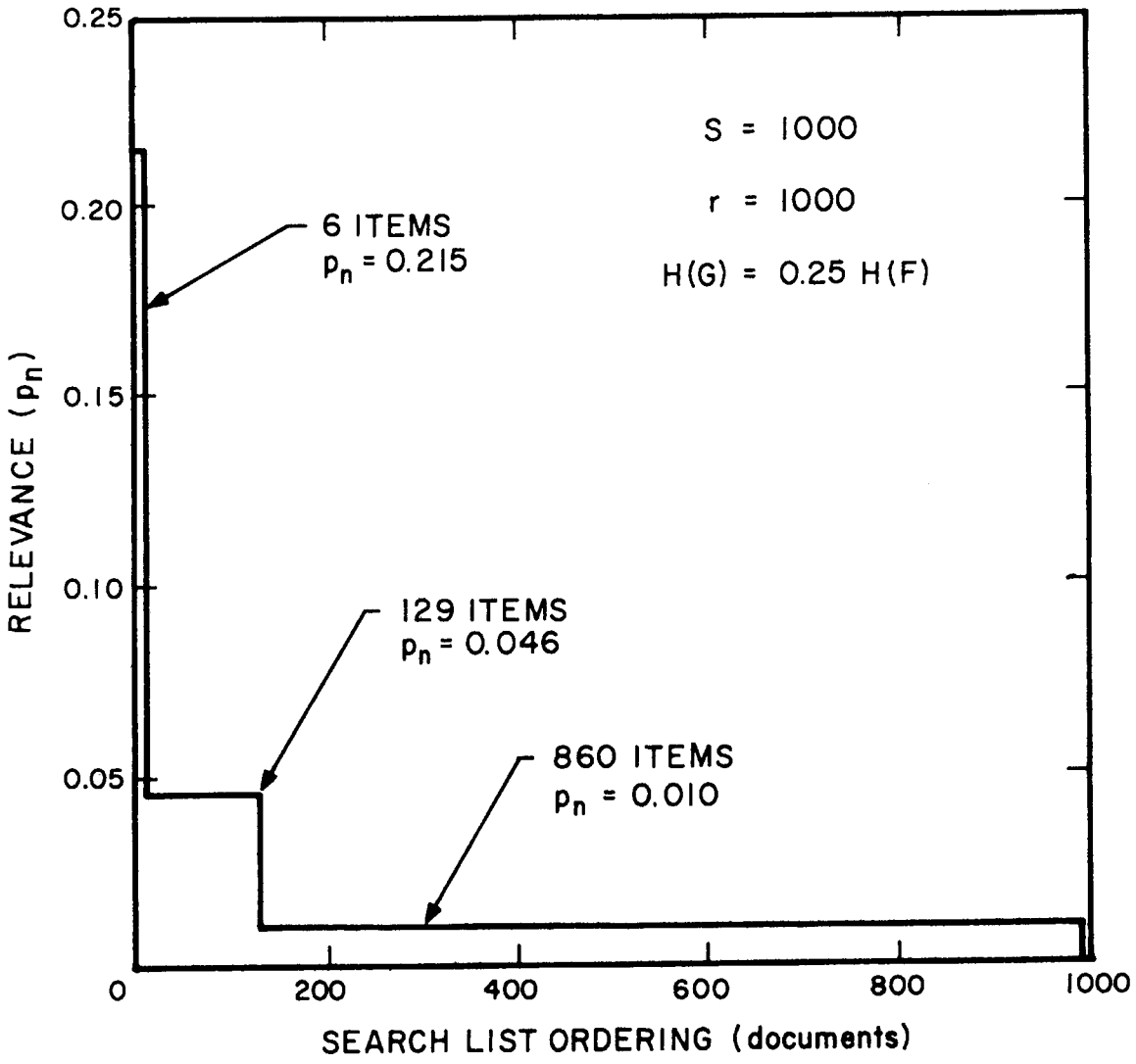

FIG. 3-6 RELEVANCE CURVE CLASSIFICATION ATTRIBUTES FOR EXAMPLE 3.3

An important observation to make is that the actual entropy figures are important for their order of magnitude, not their precision. Variations of up to a factor of two in calculation of H(E) or H(C) do not substantially decrease search lengths. It would only be if H(E) were very close in value to H(C) that search lengths would be short and precise calculations necessary.

Equivalently, variations in directory organization do not give significant improvement in search length. The file improvement techniques we must look for are those giving large increases in catalog information content, and this is not a problem of just information organization.

3.4 VARIATIONS ON THE EXAMPLE

Catalog Size. The best way to observe the influence of catalog size on search length is to calculate the total-recall search length lower bound.

$$\text{Minimum total-recall search} = s \cdot p(\#|q)^{\alpha}$$

$$\text{where } \alpha = \frac{H(E)}{H(C)}$$

For the numbers of the above example:

α	search length
0	1000 items
.05	813 items
.1	661 items
.25	356 items
.5	126 items
.8	37 items

Real Library Statistics. Estimating the size of the ideal directory for a real library is risky, of course, because it depends on the level of user interests. Care must be taken not to be biased by the statistics of existing libraries, because users always adapt their needs to the resources available. No survey of library users will turn up statistics of the questions from people who did not bother going to the library because the response would be predictably poor. Any survey of present library users will always demonstrate the need for a catalog just slightly larger than the present one, no matter how limited it is.

Therefore, the imagination must be used. Presume that the library is indexed about as thoroughly as a book is by its index. That is, the typical index is taken to be the level of classification which would approach an ideal directory. (It is not proposed that the book index be used directly, but only that the number of concepts in a book which should be identified is the same.)

Presume then a library of 10^6 items, with 100 attributes related to each item. With full indexing each attribute might be related to 1% of the items. Then:

$$s = 10^6 \qquad r = 10^4 \qquad \rho = .01$$

$$H(F) = r \cdot s \cdot H(\rho) = 8 \times 10^8 \text{ bits, or 800 bits per item.}$$

Calculating catalog implementation entropy requires some imagination as well, because it is stored in English, not binary. It is reasonable to say that a book can be accessed through four independent fields of identifiers, as was discussed in the example

necessary to substantially improve file access (50% or greater reduction).

More complicated correlations make a smaller contribution to entropy reduction. For example suppose one attribute is correlated with four others by that fact that it may appear only when one of the other four appears. For example, some subject attributes appear only in four journals, and nowhere else. The entropy of such a restricted attribute in the library example is reduced from .08 bits to .04 bits.

Correlations do appear in real libraries, but the information measure of their magnitude is perhaps smaller than one might think. An attribute must be restricted to a very narrow range to reduce its entropy noticeable: an attribute of density ρ loses half its entropy only when it has been confined to $\sqrt{\rho}$ of its original area. For example, for ρ=.01, if an attribute is known to appear in only .1 of the library, then it loses roughly half its entropy. Further, two attributes must be strongly correlated to achieve the same level of reduction. Based on these observations, I project that file entropy is seldom reduced more than 50% by correlations, and that 50% makes a good working value. Even if this 50% reduction in H(F) could be fully exploited by clever catalog construction, general search lengths would be long.

Question Correlations. A weakness in present theory of files is the lack of a means for predicting what questions a user will ask of a file. Users know about correlations between terms of their

subject field, and this clearly alters the kinds of questions asked. For example, if two attributes are disjoint (do not intersect, as for example two different dates of publication), the user will certainly not ask that both attributes be satisfied.

If notable correlations exist in Q, then the values of $p(\#|c)$ will vary widely over the various configurations. This alters the strategy in implementation of the catalog. Configurations with low $p(\#|c)$ are seldom accessed, and should have less catalog information wasted on them. Likewise if Q is biased so that some attributes are used more frequently than others, then catalog information is best spent on the most popular attributes.

The previous bound

$$\overline{-p_n \log p_n} \geq K(Q,F)[H(C)-H(E)]$$

is applicable only to unbiased Q's. With uneven use of attributes, it is not a linear relation, but one where the first few bits in H(E) are more valuable than later bits. It is difficult to generalize further, for the correlations must be treated as special cases. Frequently library policy prevents concentration of all catalog information on the most frequently used attributes and documents. In such a case, it would be a reasonable approximation to model the file with no correlations in Q for calculation of search lengths.

3.5 ORDERING BOOKS ON THE SHELF

The discussion to this point has ignored the possibility that item identification numbers may be related to item content, as when the items have been ordered by content. The previous calculations

assumed randomly selected identifiers, such as when a librarian sorts books by acquisition numbers or some such thing. In fact, the ordering of books on the shelves of a library is an alternate form of storing directory information.

For s items in a file, there are s! different shelf orderings. The entropy of these orderings is about s log s. The number of different implementations of a file is the product of the number of states of the catalog and the number of states of shelf ordering, so that the entropy of implementation is the sum of catalog entropy and ordering entropy. The two sources of entropy are indistinguishable except that the ordering entropy is limited to s log s bits.

The previous calculations considered total implementation entropy for H(G), so the introduction of ordering serves simply to reduce the number of bits in the catalog by s log s. In particular, search lengths based on ordering information alone may be obtained by setting the catalog to zero and calculating relevance curves for H(E) = log s.

Techniques. Procedures for ordering items in the file are exactly the same as those for selecting information to construct a catalog. Sample possibilities are:

1) Order items by a particular subset of attributes, as when ordering by author name, journal name, etc. This gives a family of relevance curves such that some questions are answered very well, and most questions not at all.

2) Cluster similar items into adjacent areas, such as the Dewey Decimal System does. This is equivalent to creating a set of composite attributes which are mutually disjoint (since a book can appear in only one place). This produces a two-step average relevance curve: each question has relevant items in one or two clusters with some reasonable probability, and the other relevant items are spread throughout the rest of the file. This approach gives reasonable results for partial-recall questions, with no help for total-recall questions.

3) Subdivision into smaller libraries is nothing more than creating macro attributes corresponding to the various library names, and is just a coarser version of 2) above. It also benefits partial-recall questions at the expense of total recall.

Ordering Strategy. In short, when we know how to select information for creating a catalog, we also know how to select information for ordering the file. The possible deviations to this rule are:

1) Not all information coding techniques which work in a catalog are equally useful for ordering. For the convenience of the user's remembering the code scheme, a system of disjoint attributes is best for ordering.

2) There is a different economic saving for having similar items adjacent on the shelves as opposed to having them adjacent in the card catalog. In general the time cost of

looking several places in the file is greater than the cost of looking several places in the card catalog, so shelf ordering should take place on the most frequently specified attributes, with the remainder put in the catalog. If all attributes are equally likely, there is nothing that can be done to take advantage of the cost difference.

Another observation of interest is that since shelf ordering is just an extension of the catalog, it should not use the same information. If both systems use the same information, it is just wasted in one place. A good example of violation of this rule is the procedure used on library shelves or ordering books within subjects by author's name. This duplicates card catalog information which is also by author's name, so if a user desires a particular author he can go through the catalog. It has in fact been my experience that the author-ordering of the shelves is useless for fact retrieval for this reason. A better alternative would be to order books within subject by date of publication, for this often is an important attribute not readily searched for elsewhere.

A second comment of the same nature is that card catalog subject terms too often are quite similar to ordering classification categories, so the subject information of the card catalog is partially wasted. Card subject terms can probably be chosen to be independent of ordering subject terms without loss of meaning, and with an increase of scope.

The objectives in building a catalog are three:

1) Coding efficiency - The information which appears in the catalog should be efficiently used. An objective in catalog construction is to maximize H(E) given the number of bits available for storage. This involves a) avoiding correlations between bits, b) using each bit equally likely 1 or 0, and c) have the bits represent useful information ($H(E|C)=0$). Of course, these objectives must be tempered by constraints of practical usage, such as putting information into English so it can be read.

2) Recall matching - The principal criterion in selecting cataloging mechanisms is to fit the anticipated range of user recall requirements. For the discussion here, that consists of observing whether a particular coding method favors high-recall or low-recall questions. Unlike the efficiency question of 1), there is no right or wrong here, just a range of trade-offs.

3) Indexing cost - A big cost factor in building a catalog is the human cost of extracting attribute relations from the books. I conjecture that this cost is directly proportional to the amount of information in the index data, H(E). This of course ignores practical considerations such as whose budget the indexing cost comes out of. Since indexing cost mirrors storage cost of the catalog, only the latter will be explicitly discussed, but this coding efficiency criterion should be understood to carry the weight of both costs.

In evaluating a range of cataloging methods using these criteria, we will find conflicts, for the criteria are not always independent. For example, the achievement of a shortest search-length for total recall often involves less than full utilization of catalog storage bits.

There has long been disagreement over what forms of attributes are best used for cataloging books. Generally the argument has been over which forms carried the most information or most closely approximated "natural" classifications of information. From the analysis above, it is clear that all attribute systems, implemented with equal care, contain about the same density of information. Differences between systems lies in second order considerations of relevance-curve shaping and coding efficiency.

Precision. Should attributes be short "keywords" appearing in many items (say a density of $\rho<10^{-2}$) or long phrases of more limited appearance (say a density of $\rho=10^{-3}$ or 10^{-4})? This subject has been mentioned in passing by most authors, and studied deeply by a few (5, 8). The answer, of course, is that it does not make much difference so long as the same amount of information is used in each system. Some differences are:

1) Precise terms are more conveniently coded in English, though there is less difference in a machine-language catalog. In pure information-theory coding terms, sparse terms are best coded by simply identifying them when they occur. Frequently-occurring terms ($\rho=\frac{1}{2}$) are best coded as a

binary yes or no for each item, but this is more difficult for the user to interpret and remember.

2) Greater precision terms come in greater quantity for a given total information, so that the system has more names to keep track of and recognize. This is an unknown cost factor.

3) Precise terms require a more complicated format of user questions. Low-precision terms can usually be just intersected to form user questions, but with high precision terms this yields too few combinations with which to specify a given document. With high-precision terms the user requires questions which are more powerful Boolean conditions of attributes, such as unions of intersections, or best m out of n match, etc. Only with this added flexibility does he get as much access flexibility as with simple intersections of low-precision terms.

4) Lower precision terms carry a higher portion of their information in the zero condition. Thus if users ask questions with negative attributes (ask for books which do not contain a specified attribute), low precision terms are better. However, semantic aspects seem to cause users to specify only positive associations, so high-precision terms have a slight advantage.

5) Lower precision terms tend to produce flatter relevance curves, which favor high-recall questions. Conversely, high precision terms are specified less often but with

ordering is considered to be a separate attribute. This just affects the complexity of the searching procedure.

3) Ordered attributes are achieved by ignoring certain document relationships which conflict with the partitioning. Thus they do not improve search length for total recall, as do most systems which spread a given information over a larger number of attributes. Rather, they give a relevance curve with less variance. That is, unordered attributes, fewer in number, answer some questions well and others poorly, giving a family of relevance curves of widely varying shapes. Ordering the attributes maintains the same shape relevance curve, but reduces the variation, so that all questions tend to be answered equally poorly.

4) Ordered attributes make a good system for arranging items on a shelf, which is an advantage in some cases, but also is a source of bias because some librarians feel this makes them inherently better for all applications.

Correlations. Should attributes be selected which are statistically and semantically independent, or should correlations between attributes be tolerated? In theory correlated attributes are as useful as uncorrelated ones, but in practice there are some difficulties in efficiently coding the information, especially if it is for human consumption. For example, at 50% correlation between two attributes, the way to store them is to represent one attribute conventionally by listing its occurrences, and adding one bit which says whether

or not the second attribute occurs also; when the second attribute occurs alone, it is listed directly. This system is efficient, but would take some explanation for users of a card catalog. In general, a catalog is more easily implemented if the attributes selected for it are mutually independent.

Relevance Threshold. Should attributes be considered as related to A) those items which are clearly strongly related to the concept of the attribute, or B) all those items which even hint they are related to the concept? Where is the threshold of attribute relevance? There is no absolute answer, of course, for this is a design parameter of the system.

For a given size catalog there will be room for fewer low-threshold attributes (those which count weak relevance) than high-threshold attributes (those which count only strong relevance). Low-threshold attributes produce a flatter relevance curve, which benefits high-recall questions. They also give greater variance among questions, because fewer attributes can be handled, although it is possible that this latter effect will be small (due to correlations which generally occur). Conversely, high-threshold attributes obviously give better relevance for low-recall questions.

3.7 TECHNIQUES OF DIRECTORY CONSTRUCTION

Many researchers have proposed various schemes of creating library access tools, and it is difficult to compare them. The general conclusions of the analysis here are that all such schemes

will be roughly equivalent, under equal conditions of information storage, etc. Second-order variations occur, however, and are discussed here. An important parameter in these systems is their effectiveness in coding correlated information.

Automatic Indexing. Computer extracted attribute relationships seem to give potential help in breaking the high costs of indexing. However, computer programs are unable to comprehend the full context of a word's usage as well as a human does, so automatic indexing is less accurate. This can be interpreted to mean that a computer-generated catalog must contain more raw information to contain as much useful information as a human-generated catalog, but that is reasonable on a cost basis. Thus the machine-generated catalog will contain more attributes with less information about each one. This has the previously described effect of smoothing out the variance in the relevance curve, answering all questions equally poorly rather than a few questions well and many questions badly.

As to how much extra attribute information is needed in an automatic system, that is the interesting question. This is beyond the scope of this research, but the informational terms given earlier seem to give a theoretical basis from which to pursue an answer. It is clear, however, that it may be difficult to come up with a fair comparison experiment, since the relevance curves resulting from machine generated attributes will be of different shape than those from manual indexed systems.

Fixed Vocabulary. Should indexers work within a fixed vocabulary of acceptable attributes, or should they be free to construct terms that best describe an item? This is virtually the same question as the high-threshold versus low-threshold one discussed above. Free indexing produces more attributes with stronger relevance (higher threshold), and thus gives less variable relevance curves which favor low recall questions.

Stemming. Stemming is one of a number of ways of forming a union composite attribute from several less frequent attributes (14). (It consists of reducing English words to the root stems, thus equating, say, computer, computation, and computable.) It serves to increase attribute coverage with consequent reduction in the variance of relevance curves and improvement of high-recall performance at the expense of low-recall performance. It is an interesting way, further, to handle correlations between attributes which appear as linguistic similarities, and thus it gives a partial solution to coding of correlations.

Thesaurus. A thesaurus is a set of equivalencies between attributes, so that near synonyms can be used to augment a user request (3). The thesaurus can be used in two ways: Selectively on high-recall questions only, or Non-selectively on all requests.

The Non-selectively approach is just an information reduction process whereby several attributes are unioned into a single composite attribute. It may not be stored quite that way, but that is

the effect. If the number of attributes per equivalence class is not large, then little information is saved this way, assuming that 1) the composite attribute appears in less than 10% of all items, and 2) the original attributes were uncorrelated. If the original attributes were statistically correlated in the data gathered (regardless of meaning), then the storage cost savings may be significant. If the original attributes were semantically correlated (regardless of statistics), information is actually added to the system in that the bits stored will be better interpreted, giving a better approximation to the real system. This latter effect is most likely when the thesaurus is generated from outside information, such as by an expert in the field, in which case it represents a true addition of knowledge to the file.

Using the theory developed here, it is possible to calculate what level of semantic correlation is necessary to justify including an attribute pairing in a thesaurus. For a pair of attributes of $\rho = 10^{-3}$, they need be only about 30% correlated semantically, assuming no statistical correlation, to justify combining them for a net gain in information. As the size of the equivalence class increases, each new attribute must maintain about the same correlation level to the entire previous class. For attributes with larger ρ's, greater correlation is required. Since 30% to 40% correlations are probably frequently encountered in subject attributes in technical fields, this implies that a thesaurus is worthwhile.

The effect of a thesaurus on a relevance curve is a flattening, with greater help given to high-recall questions at the expense of low-recall questions. Salton has compared systems with and without thesaurus, and shows relative precision-recall curves on page 130 of his book (3) demonstrating exactly this effect. (His curves are sufficiently similar to my relevance curves that the general patterns appear the same for both.) Calculating $p_n \log p_n$ from his numbers, his thesaurus was not able to capture a high enough correlation level to achieve an effective increase in useful information, for the two results were effectively equal. Thus his thesaurus approach shows how to reshape a relevance curve without change of system performance level.

If the thesaurus is generated statistically from the attribute data gathered for the catalog, it represents no new outside data. It still can have a net positive effect on the useful information of the catalog, but only if 1) attributes which are semantically independent also appear as statistically independent in the data, and 2) attributes which are semantically correlated also appear statistically correlated in the data. This is a very difficult set of conditions to identify quantitatively, but it looks like a tenuous justification for a non-selective thesaurus.

A selective thesaurus is another matter. In this system, equivalent attributes are employed only when the user-specified attributes fail to achieve the desired recall. This system is an addition to the information already in the catalog and cannot fail to improve retrieval. Even a statistically generated "thesaurus"

derived from statistically independent data will only serve to cause an exhaustive search over a different ordering of the items (exhaustive search being what has to be done to achieve high recall in the first place). The extent of information improvement with a thesaurus depends on semantic correlation levels (see example of correlation savings in Section 3.4). For example a 50% semantic correlation correction by a thesaurus augments the useful catalog information by about 30%, less any statistical correlation already found.

In general, it becomes clear that the thesaurus is a useful tool, and that with the correlation levels common in subject attributes, I would guess it offers savings/gain of perhaps 20-40% over the simple catalog.

Clustering. A number of researchers (10-18) have outlined proposals for statistical "clustering" methods which serve to determine correlations between attributes (or between items). Mathematically this is just an implementation method for creating the thesaurus described above. Such methods are used 1) Non-selectively to reduce data by creating classification categories having fewer attributes than the original data, or 2) Selectively upon retrieval to expand a user question to achieve fuller recall. Thus the value of clustering is covered above.

But there are many algorithms for clustering because the full problem is too big and must be solved by an approximate iterative procedure. Unfortunately, no one has found a method to evaluate

2) If raw data can be expanded, then clusters should be increased and new attributes added so long as the new data is no more unreliable than the original.

3) If no exact sizes are set (the most likely case), then clusters on n attributes are best when

$$\frac{1}{n \log n\rho} = C,$$ where ρ = density of attribute relations to documents,

and C = percentage correlation of new attributes to previous members of a cluster.

At this value of n, new attributes added to a cluster cause more error in interpreting the previous attributes than they add new information. In most practical cases (correlations of around 30%, say), this occurs when the union of documents related to the attributes in a cluster (i.e. the number of items related to any attribute in the cluster) reaches between .1 and .5 of the total document set (when the composite union attribute reaches a ρ of .1 to .5).

What density weighting factor should be used to measure cluster optimality? I have not yet found a student of clustering who recognizes this weighting factor, but it is one of the few interesting parameters in clustering. For example, consider attribute clustering: it is desirable to have each document appear (be relevant to an attribute) in as few clusters as possible. Consider an example:

Bibliographic Attributes. The source of attribute information is not a subject which can be analyzed statistically. It appears that information from any one source is as good as from any other source provided it is consistant, reliable, and understood by the user. A number of researchers have suggested bibliographic citations as an attribute set to aid in identifying subject matter (7,16,17). This seems certainly to be a useful data source to supplement subject terms, but not to replace them.

In considering various sources of attribute data, the following statistical properties might be observed:

1) Does the data source provide a capability to vary attribute density (ρ) in order to shift the level of recall emphasis? Citations, for example, have some difficulties here because book bibliographies have fixed lengths, with few inherent means to expand or abridge them. If citations are used, certainly clustering methods, such as Ivie's (17), must be avaliable to adjust ρ.

2) Does the data source contain correlations which make it inefficient to store? Before adopting a data source, some idea of the extent of correlations must be gained to indicate whether the data is worthwhile storing directly, or if it needs compaction to reduce correlation and make it economical. For citations, it is difficult to say intuitively whether or not bibliographic lists will tend to duplicate one another or have other correlations.

mechanical in nature, requiring no judgment on the part of the indexer. This is more easily said than done, but it is an interesting way to restate the goals of catalog construction.

Correlations in the Description of an Implementation. When composite attributes are used in an implementation, the zero condition of such an attribute says not only that the original attributes were all zero, but that they were zero together. This fine point turns out to cost upward to 30% in efficiency of utilization of the catalog bits. The problem is that catalog information about combinations of information is not useful because user questions tend to ask single attributes alone. This effect seems to appear to some degree whenever relevance curves are shaped for total recall, and is just an inefficiency which must be accepted whenever flat relevance curves are desired.

The principal affect on the librarian from this obscure effect is the perhaps obvious advice that when two attributes are correlated in the file, the implementation often should be chosen to break up the correlation. This is done by putting one attribute in the catalog but not the other. Then the one which is cataloged can serve to aid in accessing the other, due to the correlation between them.

Coding Efficiency. Librarians and programmers are probably all aware of the basic methods for efficient coding of directories. Little can be added except to note that, given the present system

of listing attribute names directly under each item, maximum coding efficiency is reached for attributes with lowest ρ (fewest items per attribute). This is because no efficient coding method has been found for moderate density ($\frac{1}{4} > \rho > 10^{-3}$) attributes which also permits easy access to the attribute values. Communication Theory's answer to this problem, difference coding (run-length coding) does not appear applicable.

Likewise, the coding techniques for reducing correlated information (various clustering schemes) seem adequate, given the tools available.

A disappointment, however, is the seeming failure of librarians to recognize shelf ordering of books as an extension of the catalog. This ordering is the lowest cost source of access information for the user, and should be exploited fully. Suggestions of possible change to better fit this philosophy are:

1. As mentioned above, books within decimal classifications should be ordered on something besides author name, say date of publication.
2. The decimal classification needs better explanation to the user, such as better advertising of the category definitions, or such as an in-stack thesaurus which tells the user to "see-also" other semantically adjacent categories.
3. The filing of periodicals alphabetically is a waste of ordering information, being ambiguous as well as generally useless. (Does the Journal of the Association for Computing Machinery come under: "Computing", "Association of",

"Association for", "Journal of", or "Journal for"? It takes me five minutes to find it in any good-sized library.) Grouping periodicals by subject matter would extend their usefulness. In fact, integration of periodicals into the book shelf ordering would seldom slow down the specific access of either, and would certainly help the browser.

Frequency of Utilization. Some books are accessed more often than others, and some attributes more often than others. Clearly age is the biggest factor in decreased usage, with quality of material being a smaller effect only because it is more difficult to measure. In the bound derived above, frequency of use shows up primarily in terms of the weighting on different attributes in H(E). Unfortunately the theory of what to do here is weak.

Mathematically, H(E) is best utilized when all available information is put entirely into those attributes and documents which are most heavily used, with all others being ignored. This would produce characteristics of high relevance for some low-recall questions, but low relevance for other low-recall questions and all high-recall questions. It minimizes average search length for low-recall questions, even though doing some poorly. An extreme application of this policy contradicts library policy of some access to all documents. Further, since age is a principal factor in usage, and because indexing labor is the main cost item in cataloging, the policy of concentration on most-used books implies throwing out old index cards, which actually saves nothing. In short, theory cannot add much to present practice in this area.

Non-Interactive Questions. The model assumes the user asks a question and gets an answer to that question. The results are measured relative to the question asked, and not to what the user really had in mind. The user does not interact with the catalog during its operation.

This is a two level assumption, related to both statistics and semantics. The user cannot improve the catalog performance without supplying extra information augmenting his original question. That is, a user cannot improve access for a given question through interaction. Simply put, the file search length is determined by the state of the catalog, compared to the number of states available in the catalog, and the user cannot affect this relationship.

Semantically, interaction can be quite valuable, in that the response to his original question allows the user to add more information and narrow the search. However, this semantic improvement is exactly the effect we wish to avoid in this file-organization analysis, for we must separate the statistical and semantic effects to understand each. This model studies the case of file performance within each step of an iterative session, where the user input is constant for the duration of the step.

Fixed Attribute Set. The model assumes that items are described by their relation to a set of attributes. This bothers two types of people:

1) Some people think that items should be described by their relation to each other. But observe that relationship to

another item could be considered an attribute for describing items, so this is not a different system.

2) Some people feel that fixed vocabularies cannot describe a library, because each new item will require new attributes to describe it. Again this is no contradiction, for the set of attributes is described only for items in the system, and indeed as items are added so also new attributes would be expected. The ideal catalog would simply have to include every attribute related to all its items.

In short, this aspect of the model does not cost generality.

Binary Attributes. The model assumed binary attributes for the ideal directory, with each attribute either relevant or not relevant to an item. The true case in real implementations is that two states exist but they are: 1) relevant, and 2) probably not relevant. Observe that real systems are just approximations of the ideal, and the 1,0 restriction on the ideal case causes no restriction on implementations.

A more serious problem is that multi-valued attributes must be considered. These might be quantitative attributes such as dates or book size, or attributes listed in various degrees of relatedness. The binary assumption was made for simplicity, both of derivations and examples.

The multi-valued attributed can always be recoded in binary, however, by simply representing each state of the original attribute by one binary attribute; if the original attribute had Y states it

would be replaced by Y binary attributes, somewhat intercorrelated. Further, I have seen no indication that any statistical argument used here could not be applied equally well to multi-valued attributes (with perhaps a little more complexity involved). While the assumption of binary attributes looks quite unrealistic, yet it seems to cause no loss of generality.

Intersection Question Types. The assumption that questions may consist of only attribute intersections is a necessary one, but does cause loss of generality. Better question types will have unions of intersections of attributes, or possibly best m out of n matchings.

The intersection case was studied only as a typical case, not an all-inclusive one. There is reasonable basis for hope that the results of the intersection-type analysis will apply equally well, with scale factors, for more complicated questions. The findings of Chapter Five give support for the indication that the intersection questions and best m of n type questions have very similar search problems.

Item Independence. The assumption that items occur independently in a file is not true in real cases, but probably is only slightly off. The analysis performed here could be extended to handle inter-item correlations, I think, with no change in results. The resulting arguments would be complicated to the point that they do not warrant the effort to correct a small approximation error.

This is somewhat similar to the question of coding subject terms on catalog cards. There is a fixed small set of possible subject terms used, so the terms applicable to each document could be identified by only a few characters of information. The user, however, must interpret these terms relative to all attributes which might reasonably appear on a card. Therefore, the catalog card lists the terms in English, using much more information than the few characters "minimum". The difference is just that the user is coming from a more general framework without previous knowledge of specific configurations in the file.

Ideal Directory. The concept of an ideal directory is a figment of our imagination which can never be constructed and perhaps cannot even be described with real numbers. It is used here only as as analysis tool, to permit conceptualization of otherwise difficult ideas. It is similar to a reference point used in surveying. The exact location of the reference point is not very interesting, but it acts as a tool to show the relative locations of all other points one to another. It would be nice to produce absolute statistics describing the ideal directory, but not necessary.

Breadth of Questions. The major variation in this model from some used previously is the flexibility of questions allowed the user, namely arbitrary combinations from a large number of attributes. This is based on the consideration that an ideal directory can allow nothing less. More limited sets of allowable questions are the result

Likewise, disjoint attribute sets (classifications) are easier to index than a similar number of overlapping attributes, just as the information content is lower. Further, it seems reasonable to say that the difficulty of assigning an item a spot in a classification grows as the log of the number of classes to be chosen from.

Of course these are guesses, and it would be interesting to see experiments which measure the cost of indexing documents into various attribute sets of the same information content. It would also be interesting to see how machine indexing costs compare with the reliability of the results they produce.

Performance Parameter - Search Length. The claim here is that search length, for various recall levels, is an effective single parameter measuring performance. Other parameters can be derived from this one, and will reflect conditions local to a particular implementation.

Search time, for example, is always proportional to search length, but has scale factors depending on the storage mechanism holding the file. Search time would have to include factors relating to data transfer rates, file bucket sizes, extent of parallel access, etc.

Response time, the time it takes to find an answer for a user, must reflect search time as well as the relationship between system demand and capacity. Queueing for resources is a principal component of response time, which is determined by availability of unused system

capacity, but reflects very little on directory construction except through search length.

Computation cost as a performance parameter has been ignored here, for two reasons: First, it is difficult to measure meaningfully. Second, it is not important in most retrieval systems. The latter condition arises because very little computation is performed in retrieval systems relative to the amount of storage machinery and relative to the economics of computers. The big cost item is the access mechanism in the storage device and the data channel connected to it.

3.10 CONCLUSIONS

We have performed a set of experiments with hypothetical data bases and exhaustive question sets. This form of experimentation permits a variation and isolation of file parameters not possible in mechanized experiments. The experiments have served to show the relatively small importance of some parameters, such as style of indexing, and the greater importance of such ignored parameters as catalog size and question set statistics.

The results of this work are typical of information theoretic work, being negative conclusions. We can say that certain bounds can never be exceeded, and can usually point out methods which will not approach the bounds, but cannot prove that some one technique is better than all else.

These results take the form of saying how not to carry out retrieval experiments, and that the retrieval problem will not be

solved by some simple answer. Sometimes this type of information is the most valuable.

Experimental Procedures

Much of the experimental work performed in the past has been of little value because the experiments have failed to hold enough variables fixed to learn anything. A good experiment must vary only a single parameter and observe the effects quantitatively. This has proved hard to do in retrieval experiments because many parameters were apparently unrecognized.

Catalog Size. Comparison of two different catalog philosophies is ineffective if the information in the catalog is allowed to vary as well. The worst example of this is the researcher who adds new data to the catalog (e.g. a thesaurus) and then finds that retrieval results are improved. The question of interest is not whether retrieval is improved or not, for it certainly will be, but whether such improvement is greater than if the same added information were used other ways, (e.g. more through indexing on the existing attribute set). To be effective, an experiment must show:

a) improved retrieval for a fixed size catalog, or

b) constant retrieval efficiency for a decreased catalog.

Question Set Size. Probably an even less noticed experimental parameter is the nature of the question set available to the user. If the catalog is augmented by additional attributes, the user question set is potentially enlarged, introducing some difficult experimental decisions.

1) If the user includes more information (additional attribute specifications) in his questions, this makes retrieval easier, regardless of whether the new attributes are better or worse than the old (as occurs in (16)).
2) If the user specifies the same number of attributes as before, but taken from a larger selection, retrieval will be more difficult despite the addition of directory information; but user flexibility is increased as well.
3) If the user specifies fewer attributes from a larger selection, his flexibility can be kept constant but the questions are easier for him to ask (less information required) and the answers will include more items (presumably at a loss of semantic relevance).

No one combination is best for all experiments. The important point is that the experimenter must know what parameters are varying, and must hold as many constant as possible.

Relevance Variations. Most experiments in directory construction will consist of comparing two systems having different relevance curve shapes. Comparing the results of the two approaches is inconclusive unless one system is clearly superior. The p_n log p_n measure

provides a means of comparison since it will show which system is making most efficient use of its information, measured on a common basis.

A second problem with relevance variations occurs in variation between questions within a given system. It is possible to build a low-information catalog which gives good results for a small set of questions, and this system will look like a winner in limited testing (a consideration in evaluating the Cranfield results (5)). An important experimental rule is to find out how many questions receive high-relevance responses, and use the $p_n \log p_n$ measure to compare such a system against one which yields lower relevance for a much broader range of questions. Experimentally that may be difficult, so the best useful alternative is to chose a question set which represents a true cross section of the capabilities of both systems. It is difficult to prescribe a good general approach to circumvent this form of relevance variance, but certainly it should not be ignored as has been the custom to date.

Scaling. A most important aspect of retrieval experiments is the possibility of performing small experiments to gain information about big systems. What is the reliability of data gained from a small data base as extended to a large data base?

1) Statistical Variance - Of course the small data set is more likely to have extreme combinations of data which give unreliable results. Any experimenter working with limited

data must make careful statistical checks of his reliability, or must edit his data to exclude extreme conditions.

2) Attribute Set - As a set of documents grows, its decribing attribute set will grow as well, but probably not linearly. Catalog methods which are sensitive to the size of the attribute set must be viewed with suspicion unless well analyzed. For example, attribute clustering algorithms may be presumed to require computation time growing with the square of number of attributes. Other systems which store combinations of attributes will find square-law growth of storage requirements (i.e. combinatorial schemes (9)). It is probably true that a thesaurus or other inter-attribute linkage system would have to grow at least linearly and sometimes as the square of attribute numbers.

3) Questioner Needs - The questioner will need to supply more information if he is to keep his search length constant as document set grows. Thus question sets will change, altering the statistics of retrieval. Any system which is geared to a given question precision will have difficulties being scaled, (again a problem in (9)). The amount of information needed to form a question grows as $\log_2 s$, so perhaps the full attribute set will grow at this same rate.

If user questions are constrained to stay constant as the library grows, then user recall levels must drop or larger searches encountered (which the user presumably pays

for in some way). This factor is often overlooked by measurement techniques which give relevance and recall only (3), while hiding the physically important parameter of search length.

Cranfield Project Results

The Cranfield Research Project (5) is worth criticizing specifically because, in general, it was a well thought-out and executed set of experiments. However, when the criteria of this paper are considered, the Cranfield work is seen to have still a few weaknesses which cast doubt on the results.

The principal problem is that catalog size was not held constant. Details of catalog sizes are not available in the published reports, but estimates can be made in places. A principal conclusion of the report is that "simple concept" terms (language II) are consistently inferior to "single word" terms (language I). The former are multi-word terms having greater precision (lower ρ). It seems, however, that fewer "simple concept" terms were used than "single word" terms, whereas the reverse would be necessary to maintain constant catalog size. I estimate that language I had 50% more information in its directory, so of course it did better. Other similar comparisons between languages having differing catalog sizes always showed the results we would expect.

Other criticisms are of style rather than technique. The final language rating system used, "normalised recall", is seen to be a weighted average p_n with higher weights for low recall levels.

This rather arbitrary measure seems to assume a linear distribution of user recall levels. No notice is taken of the fact that other recall distributions might be more likely in real situations. It is not that I do not like the measure used; it is that I object to it being considered superior to all other schemes of arbitrarily weighting recall levels.

Likewise, the recall-precision curves used here (and by Salton) are sometimes misleading in their presentation of data because they smooth out important fluctuations. If these curves were differentiated with respect to recall, they would show "precision" (p_n) for the added items needed to achieve the next level of recall. For example, in many of the curves shown, the increment from 80% to 90% recall was done at essentially exhaustive search (i.e. picking items at random would do as well), but this is not easily seen as presented. They also give undue visual weight to the relevance of the first few documents retrieved; if the first document retrieved is irrelevant, it affects the whole slope of the curve.

On another point, it was observed that the Cranfield investigators thought it interesting that relevance ("precision") decreased with increasing recall, on average. It should be obvious that there is no other possibility. If the search system has some information about the relative p_n's of the documents it must search, the most probable will be accessed first. If no p_n information is available, documents will be accessed randomly, giving a flat precision-recall curve. Nothing can possibly give a precision-recall curve with positive slope.

These factors, plus the fact that many of the interesting experiments were run on a small data base, lead me to distrust the Cranfield results. Their approach and execution were good quality, but the problem is a difficult one.

In general, the important aspect of experimental work is in defining the parameters of the system and their relative importance. The theoretical work done here provides a tool for analysis of the experiments, but more important it shows exactly which parameters affect search length. This identification seems overdue in the retrieval field.

Strategy for Library Construction

The basic result of this work is that library effectiveness is related to the amount of information in the catalog first, and all other aspects a distant second. If the library is to become an effective source of information, then, all that is needed is to have huge catalogs. There are perhaps a few objections to this approach, however:

1) Catalog Search Time - As the catalog approaches having as much information as the main file, the problem of accessing the catalog will become as great as accessing the file originally. To overcome this, deliberate redundancy must be introduced into the catalog so that all varieties of questions can be responded to with low search times. However, this implies a catalog whose size far

Catering to low-recall questions is the only way a library can use limited catalog resources to satisfy a wide range of questions.

Indeed, in an interactive library, response to low-recall questions must be emphasized because interaction will encourage users to explore system semantics through numerous investigative questions. Perhaps 95% of all questions will be low-recall in such systems. This implies the need for a very large number of low-density attributes, probably generated by authors.

CHAPTER 4

MATHEMATICAL ANALYSIS OF FILE DIRECTORIES

This mathematical analysis of directories employs two conceptual tools of some interest. First, it defines an ideal directory against which all real implementations can be compared. Second, it defines an ensemble of directories, reflecting the various configurations of file data which a file user might expect to encounter. Then various directory construction schemes can be compared as they perform on a given ensemble.

A general lower bound on search lengths is constructed, based on the difference between the information in the ideal directory and the information in a real implementation. The interpretation of this bound serves as a guide in constructing directories and in measuring their performance.

4.1 DIRECTORY SYSTEM MODEL

An ideal directory f is defined as a binary matrix having:

r attributes (columns)	$a_1 \ldots a_i \ldots a_r$
s items (rows)	$b_1 \ldots b_n \ldots b_s$
a binary relationship	$f(a_i\ b_n) = 1$ or 0 .

A question ensemble Q is a set of 3^r questions q, each composed of:

an r-bit vector of attribute values, each specified to be 1, 0, or $\emptyset$ (don't care).

a probability of occurrence $p_Q(q)$.

Together, Q and f define a file, where an item b_n is said to be relevant to a question q iff for every a_i:

$$f(a_i\ b_n) = 1 \ \& \ q(a_i) = 1 \text{ or } \emptyset$$

$$\text{or } f(a_i\ b_n) = 0 \ \& \ q(a_i) = 0 \text{ or } \emptyset \ .$$

Notes. This is the intersection form of question, where an item must fit all specified attribute values of q to be considered relevant, and where both 1 and 0 in f fit a $\emptyset$ in q.

The directory is ideal in the sense that the user agrees with the evaluations of relevance so defined (which assumes that all semantic problems have been overcome).

The use of f is that whenever an item b_n is found relevant to a user question q, the item identifier n is returned to the user to identify the relevant item in the main file.

The ordering of rows of f is determined by the main file, and this is assumed to be not correlated with the attribute information.

One restrictive assumption is made here, namely that the items of a file are statistically independent of each other. This permits an item by item analysis, which reduces the notational complexity in the following analysis. This assumption does not affect generality of the results, for the derivations could have been carried out without it, with a loss of clarity.

Item Configurations. An item can take on a binary configuration c with probability $p_C(c)$. This defines the ensemble of configurations

C, which has 2^r members. These probabilities are defined by the user's knowledge of possible file contents, not by a particular file. The user's uncertainty about an item's configuration is measured by:

$$H(C) = - \sum_C p_C(c) \log^* p_C(c) \qquad (4.1)$$

The maximum value of H(C) is r, which occurs when attributes take on values 1 and 0 equally probably and independently within C. When the attributes have known correlations or unequal 1, 0 probabilities, $H(C) < r$ results.

The ideal directory is just s samples from H(C), so this defines an ensemble F of directories (files), and an entropy:

$$H(F) = s \cdot H(C) \leq r \cdot s \qquad (4.2)$$

Implementation. An ideal item configuration c is represented in the real world by a representation e. There exists an ensemble E of these e's with each member having a probability of occurrence $p_E(e)$. Thus an entropy H(E) is defined (as in 4.1), which in general will be smaller than H(C). The probability of occurrence $p_E(e)$ is based on the probabilities of occurrence of the c's it represents. In some cases, several e's may be alternate representations for one c, so exact statements about H(E) must await the study of special cases.

---

* Throughout this paper, except where noted, log x means $\log_2 x$.

The representation of an f is defined to be a g, which with its probability $p_G(g)$ is a member of the ensemble G; $H(G) = s \cdot H(E)$. The physical meaning of H(G) is straightforward, being the number of bits of information used to construct the file directory. If the directory is a table of values, this bit count is obvious. If it is a hash-code algorithm, the number of bits in the stored program (which determines how many different algorithms could be used) is H(G).

Whenever a particular representation e occurs, this means that one of the c's it represents would have occurred had the implementation been an ideal directory. For each such e, a conditional probability $p(c|e)$ is therefore meaningful. Then $H(C|e)$ is the entropy of these c's, and:

$$H(C,E) = H(E) + \sum_E p(e)\, H(C|e) = H(E) + H(C|E) \qquad (4.3)$$

4.2 TOTAL-RECALL EXAMPLE

To illustrate the nature and meaning of the derivations constructed in this chapter, a simple example is examined first. It has restrictions on the question set and on implementation techniques allowed, to simplify the mathematics. While the restrictions may seem harsh, the model does actually fit a number of interesting situations.

1. Questions:
   a) Total Recall - every user question is assumed to require total recall, so every item which might possibly fit the question is retrieved.

b) Attribute Values - every attribute has the same fixed probability of being specified 1 or 0, and all attribute values are specified independently, so:

$$p_Q(a_i=1) = p_Q(a_i=0) = \frac{1}{2} p_a$$
$$p_Q(a_i=\emptyset) = p_\emptyset = 1-p_a \tag{4.4}$$

2. Implementations

a) Partition - Only implementations are considered which partition the set F, so that any one c is always represented by the same e, and

$$H(C|E) = 0. \tag{4.5}$$

b) Representation - only those implementations are considered which can be represented by an r by s matrix whose elements are 1's, 0's, and ∅'s.

For examples of implementation which fit these restrictions, consider two r=2 implementations:

A. One attribute can be just ignored (with an obvious saving of storage space), so that, say c_2=10 and c_3=11 are stored as 1∅=e_1, and c_0=00 and c_1=01 are stored as 0∅=e_2.

B. The two attributes can be combined, such as:

c_0 = 00 is stored as 00 = e_1

c_1 = 01 is stored as 01 = e_2

c_2 = 10 and c_3 = 11 are stored as 1∅ = e_3

(three configurations are cheaper to store than four).

Items are retrieved whenever the question fits the representation e, with a $\emptyset$ in e being interpreted to fit both a 1 and 0 in q. In example B immediately above, if q = 11 is asked of a file represented by e_3, the item involved will be retrieved whether its true identify is 11 or 10.

Now we consider the probability of relevance of a particular item b_n when it is retrieved through a particular representation e. The probability of relevance, p_n, is the probability that the item, when retrieved, actually fits the question asked, averaged over all questions and all configurations:

If e contains no $\emptyset$'s, $p_n = 1$ (or else it is not retrieved)

If e contains one $\emptyset$, in attribute a_i,

$p_n = 1$ if $a_i(q) = \emptyset$

$p_n = p_C(a_i=1)$ if $a_i(q) = 1$

$p_n = p_C(a_i=0)$ if $a_i(q) = 0$

$$\text{Avg.}(p_n) = 1 \cdot p_\emptyset + p_a\left[\frac{1}{2} p_C(a_i=1) + \frac{1}{2} p_C(a_i=0)\right]$$

$$= p_\emptyset + \frac{1}{2} p_a = 1 - \frac{1}{2} p_a$$

If e contains x $\emptyset$'s,

$$\text{Avg.}(p_n) = \left(1 - \frac{1}{2} p_a\right)^x$$

It is more convenient to deal with $\log p_n$, so if e has x $\emptyset$'s:

$$\log \overline{p_n}(e) = x \log \left(1 - \frac{1}{2} p_a\right) \qquad (4.6)$$

But now observe that $x \geq H(C|e)$, because with x $\emptyset$'s in e there are at most 2^x c's represented by e; if these are equally probable, then $H(C|e) = x$, but if they are unevenly distributed $H(C|e) < x$ (because $\log \overline{z} \geq \overline{\log z}$). Therefore, from 4.6:

This gives a definite upper bound on $\overline{p_n}$. It also helps describe the trade-offs available in deciding how to allocate directory information among attributes and items. A principal result here is that the error level in the directory performance is linear with the difference between H(F) and H(G). Further discussion of this relation appears later in this chapter and Chapter Three.

In summary, the derivation consists of

1) Observing how many file configurations are represented by a single implementation state;
2) Observing that an item must be considered for retrieval if any of these possible configurations would fit a given question,
3) Calculating how often the item is erroneously retrieved because the question fit one configuration under the specified representation, but another had actually occurred.

The latter calculations are very dependent on the statistics of the question ensemble, Q, especially to the extent that Q distinguishes the various configurations (c's) represented by a single implementation state (e). The above line of reasoning is developed below with greater generality.

4.3 GENERAL BOUND DERIVATION

For the general directory analysis, we are given:

Q, a question ensemble; $q \in Q$

C, an item configuration ensemble; $c \in C$

E, an item representation ensemble; $e \in E$.

Definitions:

is the symbol for the event that the configuration c fits q, the question at hand. (# is pronounced "fit".)

p_n is the probability that item b_n is actually relevant to q, within a given implementation:

$$p_n = p(\#|q,e) = \sum_C p(\#|q,c)p(c|e), \tag{4.10}$$

$$\text{where } p(\#|q,c) = 1 \text{ or } 0. \tag{4.11}$$

Measure to be Evaluated:

The average of log p_n:

$$\overline{p_n \log p_n} \doteq \sum_Q \sum_E p(q)\ p(e)\ p_n \log p_n \tag{4.12}$$

This measure is not picked for reasons of any hidden significance but rather because 1) it can be evaluated mathematically, and 2) it gives interesting results. Other weightings of p_n might prove more interesting, but probably are much harder to evaluate.

Theorem:

$$-\overline{p_n \log p_n} = I(Q\#;C|E) \tag{4.13}$$

(The information term is defined by

$$I(X;Y) = H(X) - H(X|Y), \tag{4.14}$$

and is just a short form for expressing differences in entropies.)

Proof: The strategy of the proof is straightforward, in just expanding p_n and reducing terms when possible. The tactics are a little trickier.

To evaluate $\overline{p_n \log p_n}$, we note that

$$p_n = p(\#|qe) = \frac{p(\#q|e)}{p(q|e)} \quad , \tag{4.15}$$

using the identity $p(x,y) = p(x) \cdot p(y|x)$ (4.16)

It is an assumption of this model that question statistics are fixed, and not adjusted by the user to reflect knowledge of the actual implementation. By this assumption:

$$p(q|e) = p(q) \quad . \tag{4.17}$$

Then using 4.17 to restate 4.15, and averaging over Q and E:

$$\overline{p_n \log p_n} = \sum_Q \sum_E p(q)\ p(e)\ p(\#|qe) \log \frac{p(\#q|e)}{p(q)} \tag{4.18}$$

Now this expression is also valid when E = C, because C is one possible implementation. Then 4.18 interpreted for C can be subtracted from the original 4.18, and the p(q) terms cancel out:

$$\overline{p_n \log p_n} - \overline{p_n(c) \log p_n(c)} = \sum_Q \sum_E \sum_C p(qec\#) \log \frac{p(\#q|e)}{p(\#q|c)}. \tag{4.19}$$

But $p_n(c) = 1$ or 0 by 4.11, so

$$p_n(c) \log p_n(c) = 0, \tag{4.20}$$

so this term vanishes from 4.19. By the definition of the entropy function, used on both log terms of 4.19:

$$-\overline{p_n \log p_n} = H(Q\#|E) - H(Q\#|C) \tag{4.21}$$

But $H(Q\#|C) = H(Q\#|EC)$ because the Q# values are dependent on C and independent of E. Then 4.21 can be reduced by 4.14 to 4.13.

QED

4.4 A WORKING APPROXIMATION (RIGHT-HAND SIDE)

As it stands, the right-hand side of 4.12 is intelligible only to information theorists. For many file cases, it is quite difficult to obtain an exact expression for $I(Q\#;C|E)$ because correlations between the terms cause severe complications. However, a general form can be developed which gives insight into most cases. This shows in essence:

$$-\overline{p_n \log p_n} \geq K[H(C) - H(E)] \ , \tag{4.22}$$

where K is a constant not dependent on E, and equality occurs when:

$$H(E|C) = 0 \ , \text{ and}$$
$$H(C|E) = \sum_i H_C(a_i|E) \tag{4.23}$$

(a restriction on the correlation between attributes in E).

To show this:

$$I(Q\#;C|E) = H(C|E) - H(C|EQ\#) \text{ by 4.13} \tag{4.24}$$

$$H(C|E) = H(a_1 \text{ in } C|E) + H(a_2 \text{ in } C|E\ a_1)$$
$$+ H(a_3 \text{ in } C|E\ a_1a_2) + \ldots \tag{4.25}$$

$$H(C|EQ\#) = H(a_1 \text{ in } C|EQ\#) + H(a_2 \text{ in } C|EQ\#\ a_1)$$
$$+ H(a_3 \text{ in } C|EQ\#\ a_1a_2) + \ldots \tag{4.26}$$

This decomposition is based on simple probability manipulations, as in Gallager (19), page 22. We compare the two expressions term by term. As an example consider the second term of 4.26, expanded on the possible conditions of a_2 in Q:

$$H(a_2 \text{ in } C|EQ\#\ a_1) =$$

$$\sum_E p(E)[N_2(e)]^{-1}[p_Q(a_2=1)H(a_2 \text{ in } C|e\#a_1,Q \text{ with } a_2=1)$$

$$+ p_Q(a_2=0)H(a_2 \text{ in } C|e\#a_1,Q \text{ with } a_2=0)$$

$$+ p_Q(a_2=\emptyset)H(a_2 \text{ in } C|e\#a_1,Q \text{ with } a_2=\emptyset)]. \qquad (4.27)$$

where $$N_2(e) = p_Q(a_2=\emptyset) + p_Q(a_2=1)p_e(a_2=1) + p_Q(a_2=0)p_e(a_2=0)\ . \qquad (4.28)$$

The first two terms of 4.27 are zero because a_2 in c must have the same value as a_2 in q if c fits q. Thus, for example, when q is known to have $a_2 = 1$, and c is known to fit q, then $a_2 = 1$ in c, and there is no entropy.

The third term requires deeper analysis. To this end, a notational simplification is introduced:

$$H(a_2 \text{ in } C|e) = H_e(a_2) \qquad (4.29)$$

That is, H_e describes an entropy within the c's represented by e. Then 4.27 can be rewritten:

$$H_E(a_2|\#Qa_1) = \sum_E p(e)[N_2(e)]^{-1}p_Q(a_2=\emptyset)H_e(a_2|a_1\#,Q \text{ with } a_2=\emptyset)\ . \qquad (4.30)$$

Now this entropy term can be bounded:

$$H_e(a_2|a_1\#,Q \text{ with } a_2=\emptyset) \leq H_e(a_2|a_1\#,a_2=\emptyset \text{ in } Q) \qquad (4.31)$$

(This follows from the inequality $H(X|y) \leq H(X)$, which is a consequence of $\log \overline{x} \geq \overline{\log x}$.)

Equality in 4.31 occurs when $H(a_2|a_1\#q)$ is the same for every q having $a_2=\emptyset$. This condition can be reached by making a_1a_2 independent of the remaining attributes. For a demonstration of this, consider the relation of a_1a_2 to a_3:

$$\text{if } H_e(a_2|a_1a_3) = H_e(a_2|a_1) \ , \tag{4.32}$$

then for the question q_1 having a 1 for a_3 and $\emptyset$'s elsewhere:

$$H_e(a_2|a_1\#q_1) = H_e(a_2|a_1,a_3=1) = H_e(a_2|a_1) \ . \tag{4.33}$$

Extending this reasoning to other q's shows that independence between a_1a_2 and the other attributes in e produces for every q having $a_2=\emptyset$:

$$H_e(a_2|a_1\#q) = H_e(a_2|a_1) = H_e(a_2|a_1\#,Q \text{ with } a_2=\emptyset) \tag{4.34}$$

Thus, the independence condition is sufficient for equality in 4.31. Full independence between the attributes in e is not necessary for equality, however, if Q is so badly correlated as to not test some attribute combinations, but these degenerate Q's are not of interest here. The conditions of 4.34 also occur, regardless of E, when $p_Q(a_i=0) = p_Q(a_i=1)$ for all a_i.

It is not correct to substitute 4.34 into 4.31, so a firm bound on 4.30 is not directly available this way. However, the reasoning involved indicates that attribute independence within e is a desirable design goal.

For all other conditions, 4.30 is difficult to analyze, so that the strategy used here is to complete the analysis using attribute independence in e, and then discuss possible variations.

Under the conditions of 4.34, 4.27 reduces to:

$$H(a_2 \text{ in } C|EQ\#a_1) \leq \sum_E p(e)[N_2(e)]^{-1} p_Q(a_2=\emptyset) H_e(a_1|a_2) \qquad (4.35)$$

Now the $N_2(e)$ terms provide a little trouble because it is necessary to average their reciprocals. As seen by their definition, 4.28, these terms are essentially independent of e under two conditions:

1) if $p_Q(a=\emptyset)$ is close to 1, it determines N,

2) if $p_Q(a=1) = p_Q(a=0)$, then N is fixed.

In virtually all file conditions of interest, one or both of these conditions is true, and the variation in $N_2(e)$ from the average N_2 is quite small. In general, variability in $N_2(e)$ does not interfere with the inequality, but the demonstration of this is not worth the difficulty. Ignoring the variation in N_2, 4.35 reduces to (using 4.29):

$$H(a_2 \text{ in } C|EQ\#a_1) \leq N_2^{-1} p_Q(a_2=\emptyset) \cdot H(a_2 \text{ in } C|Ea_1) \qquad (4.37)$$

Returning to 4.26, and performing the same process on each term:

$$H(C|EQ\#) = N_1^{-1} p_Q(a_1=\emptyset) H(a_1 \text{ in } C|E)$$

$$+ N_2^{-1} p_Q(a_2=\emptyset) H(a_2 \text{ in } C|E\, a_1) + \ldots \qquad (4.38)$$

If the statistics of p_Q are the same for all attributes, then the N_i are constant over i, as are the $p_Q(a_i=\emptyset)$. Then using 4.25:

$$H(C|EQ\#) \leq N_i^{-1} p_Q(a_i=\emptyset) H(C|E), \text{ and} \qquad (4.39)$$

$$I(Q\#;C|E) \geq \frac{N_i - p_Q(a_i=\emptyset)}{N_i} H(C|E), \text{ from 4.24.} \qquad (4.40)$$

are necessary, of the form:

$$H(C') \leq \sum_i W_i \, H(a_i \text{ in } C) \tag{4.44}$$

where the W_i are weights for the various attributes. When H(C') and H(E') are thus defined and substituted into 4.42, the bound holds for the more general case. For discussion purposes, the normal H(C) is used below, but H(C') should be substituted for cases where it makes a difference.

Now the theorem 4.13 can be used to arrive at:

$$-\overline{p_n \log p_n} \geq K \cdot [H(C) - H(E)] \tag{4.45}$$

The constant K can be easily evaluated by the observation that if $H(E) = 0$, $p_n = p(\#|q)$:

$$-\sum_Q p(\#|q) \log p(\#|q) \geq K \cdot H(C) \tag{4.46}$$

The bound can be restated in alternate form (equally useful) by summing over all items:

$$-\sum_n^s p_n \log p_n \geq K[H(F) - H(G)], \tag{4.47}$$

This is the working version used for the application discussions.

Interpretations. The principal value of this bound is its quantification of file access efficiency relative to directory size. In addition, several second-order effects, due to assumptions and inequalities used, indicate methods which should be used to minimize the size of the access error. These aspects of selecting E are discussed here:

3) Unequal Usage of Attributes. The conditions which caused introduction of the weighted entropies H(C') and H(E') cause some attributes to be more important than others. To minimize $H(C'|E')$ for a fixed $H(C|E)$, the information in H(E) should be concentrated in the more heavily weighted attributes.

4.5 RETRIEVAL BOUNDS (LEFT HAND SIDE)

For a given library system, when we have determined an information level for the bounding relation, 4.47, retrieval performance can be bounded by the $\overline{p_n \log p_n}$ term. The bound says that $-\overline{p_n \log p_n}$ must be larger than the information level determined by the design parameter of directory size. Large values of $-\overline{p_n \log p_n}$ can be achieved only by making individual p_n's smaller. Thus, a smaller directory (smaller H(G)) causes a larger $I(Q\#;C|E)$, causing smaller p_n's, which means that relevance for each file access is reduced. However, there is some latitude in the form in which this decreased relevance appears.

The critical parameter is how the relevance error is distributed amongst the various items, b_n's. First observe that:

$$\sum_n \sum_E p(e)\, p_n = \text{constant} = s \cdot p(\#|q) \qquad (4.59)$$

since $p_n = p(\#|qe)$

That is, the number of items in the file which fit q is the same regardless of how the file is organized. Due to file organization, the various p_n's may be adjusted, reflecting uncertainty

about the exact identity of the relevant items, but the summed p_n must always give the same expected value. The strategies for dividing up $p(\#|q)$ are analyzed here according to desired retrieval performance characteristics.

Total Recall. If every question requires total recall, then every item with non-zero p_n must be accessed. To achieve maximum $\overline{-p_n \log p_n}$ with fewest non-zero p_n's, all non-zero p_n's should have the same value (equivalent to choosing a message ensemble of fewest messages for fixed entropy). That is, there would be X items in the file with $p_n = \frac{s \cdot p(\#|q)}{X}$, and s-X items with $p_n = 0$. While this distribution is difficult to achieve in practice, it gives a rigid lower bound on full-recall search length.

Note that when $H(E) = 0$, $K \cdot H(C) = -p(\#)\log p(\#)$, so $-p_n \log p_n \geq p(\#)\log p(\#)$. The only solution to this is $p_n = p(\#)$, and $X = s$, indicating exhaustive search of the file is required, as we would expect.

To achieve a flat p_n distribution, the information in the catalog G must be evenly taken from all attributes in F. This means that every question asked gets about the same amount of information from G. Further, the information must be evenly related to all items, so that no items are left to access by chance. All of this is achieved if the information reduction by which G is produced from F is achieved by unioning F's attributes into new composite attributes in G. That is, G will be implemented with many

Mixed Recall. For many applications, especially libraries, users do not always want all file items related to their question, but want "one good book", or a few references, etc. The question ensemble must include, then, some description of the recall levels prescribed by the users. For example, each level of recall might be equally likely, so that a file system must be prepared to serve a full range of recalls.

For mixed recall, the $p(\#|q)$ can be redistributed among the p_n such that some p_n are not too small, while many p_n are quite small. The items with large p_n get accessed for short-recall questions, giving very short searches in those cases. However, total recall searches are extended, sometimes to exhaustive search of the file. This form of directory is best accomplished by creating G with a random (or pseudo-random) sampling of data from F. Thus, for any one question, there will be a few items which are completely described (relative to that question), and give very good p_n's. Most items will have fewer descriptive bits relative to q in the catalog, yielding poor p_n's. Exact analysis of resulting average search lengths has not been attempted, but the results would still not give short search lengths.

Favored Questions. A third approach is to give very accurate answers for a few select questions, at the cost of very poor search lengths for most questions. In systems of this sort, users who wish to ask the questions for which poor response will be obtained just take their questions elsewhere, and this greatly reduces average search lengths.

This selective responsiveness is achieved by having G contain just a subset of the attributes of F. Thus, questions which are stated in terms of the selected attributes will get very good service, and other questions will fare less well. Unfortunately, for H(G) less than half H(F), the number of well-served questions is quite small, often on the order of a few percent.

The above three variations on p_n distributions summarize all options available. Indexing and coding schemes used in practice yield some combination of these three effects.

The most important aspect of this discussion of coding techniques is that generally the variations achievable in average search length are second order ones. The first order effect in reducing search length is the size of H(G) relative to H(F). Search lengths can get quite small only when G contains nearly as much information as F.

4.6 ORDERING

All the previous derivations assumed that the main file was ordered (had addresses assigned) by some criterion other than attribute values. If the file can be arbitrarily reordered, then for any single state in the directory, there can be s! different states in the file. Thus for each bit configuration of the directory, there are s! g's in G. Then

$$H(G) \leq \log s! + H(\text{directory}),$$

where equality is reached if the information in the directory is uncorrelated with the ordering information.

Equivalently, it can be observed that in a file ordered on attributes, any particular file location b_n will take on about 1/s as many item configurations as before ordering, so

$$H(C|E) \geq H(C|E \text{ unordered}) - \log_2 s.$$

The conclusion of this is that ordering is part of the implementation, and H(G) includes ordering information. Thus the criteria for selecting information to be stored in the directory apply as well to selecting ordering algorithms.

CHAPTER FIVE

COMMENTS ON FILES IN GENERAL

The previous analysis has assumed a directory-type system with the main file stored on an item-by-item basis. Are there other methods of file storage which give shorter searches for equal storage costs? For specific cases yes, but in general no. But that remains to be shown.

5.1 THE GENERAL FILE SEARCH LENGTH PROBLEM

Given a file F of s items and r attributes, and a question ensemble Q, what is the minimum average number of bits which must be examined in any implementation to find the items of F which fit each question?

This is an open problem. The best that has been done to date is some limited bounds on special cases. The information available from special cases strongly supports the conjecture that presently-used simple file organizations give near minimum search lengths.

I feel that a theory can be developed analyzing the limits of usefulness of each bit of an implementation, thereby bounding the number of bits necessarily examined for each question. This might be done by the observations:

1) Each bit must be equally likely 1 or 0 for efficient catalog construction; it can be shown that alternatives always give longer searches.

2) The 1 condition must represent the occurrence of a set of x attribute configurations for a set of y documents; if x is large, then y is small, and vice versa.
3) For each 1 occurrence, further information must be examined to identify which document within the set of y is being described, and which configuration within the set of x has occurred.
4) Due to the logrithmic cost of state storage, it is better to work at the extreme values: one configuration over many documents or many configurations for one document. Thus, most efficient storage occurs for conventional item-by-item storage, or inverted files, but not in between.

A few holes exist in this argument so a proof cannot be claimed, but a few clever insights will probably produce a proof of this nature someday.

Presuming the argument is correct in objective, then it suffices to study only item-oriented files or inverted files. All present evidence indicates that if one example is to be studied, the item-by-item organization is the best such example.

No-Directory Analysis. The Appendix develops a combinatorial analysis of directoryless item files. It gives a lower bound on search length regardless of the ordering of the items. It gives a means of predicting full-recall search lengths as attribute set size (r) varies. It also provides a tool for exploring the effects of storage redundancy on file access.

If this combinatorial analysis is compared with the previous entropy analysis, for the case where $H(G) = s \log s$, it is found that the entropy analysis gives a much tighter bound on search lengths. This is because of the very poor approximations made for distribution of pair-distances in buckets in the combinatorial analysis of search length for arbitrary organization.

Despite its looseness, the combinatorial bound does show that search lengths approach the full file length as r becomes large with respect to t (t = number of attributes specified in a typical question). Both systems thus agree that as the user demands increased flexibility in accessing a file, his average search length must increase.

Redundancy. The most important use of the combinatorial approach in the Appendix is in analysis of file redundancy. That is, if items are repeated in appearance in the file, how does this improve search lengths? This question is difficult to investigate by any means other than the combinatorial one.

That analysis says that the lower bound on search length decreases linearly with redundancy. That is, if each book in a library appears twice, search length is halved. Further, the error in the combinatorial lower bound appears to be roughly constant over various levels of redundancy, so that the theoretical linear inverse relationship should carry over into practical cases. The simple examples which can be calculated tend to confirm this.

The combinatorial analysis, however, was performed for a very limited set of conditions, namely $\rho = \frac{1}{2}$ attributes and no correlations in either questions or attributes. There is no reason to expect the results to differ in general for other systems, but the possibility exists.

The strong implication of the above reasoning is that there is no way to store information efficiently in a file to give short search lengths for large r. Further, redundant coding in the file gives only linear improvement in search lengths, which is little help when these lengths were very long initially. The only possibility left is that a directory might be of benefit if it is made big enough.

5.2 DIRECTORY FUNCTIONS

The problem with a directory is that it is a file and it too must be accessed. If it is as large as the original file, then certainly the directory search problem is no smaller than the original file search problem. There are two reasons to believe that putting information into a directory serves only to shift the search problem from the main file to the directory:

1) The bound derived in Chapter Four shows a linearity in the utilization of H(G), provided all the bits of H(F) are used equally often. That means that no one bit in H(G) is more important than any other bit, so there is no informational advantage for putting the bits in the directory.

2) The vaguely defined line of proof in Section 5.1 would be applicable to bit patterns no matter how they are organized, so information in a directory is seen as just a recoding of file data. But such recoding cannot reduce the number of bits which must be accessed.

A confusing aspect of directory analysis is the role of item ordering in the main file. If the items are ordered according to attribute information, this serves to reduce the demands on the directory by log s! bits. That analysis leaves open the question of whether or not directories can be cascaded, with each level of a directory to a directory gaining log s! bits.

No such gain is possible. The previous analysis neglected to point out that when H(F) was unordered, the catalog was required to store file addresses in addition to attribute information. The ideal directory then theoretically must store H(F) + log s! bits, but this was ignored because the directory can itself be ordered to reduce the storage requirements by log s! bits. Then when the main file is ordered, the gain of log s! in the directory occurs only if the directory remains ordered itself. But this means that the directory order is fixed and no flexibility is left to effect another log s! savings.

So the directory does not really play a role in utilizing the ordering information, but it has other practical uses.

Advantages of Directories.

1) Redundancy - Generally, information in a directory augments file information rather than replacing it as it might theoretically do. Thus a directory is a form of redundancy which does indeed reduce search lengths. But search lengths are at best linearly reduced with redundancy, so for a substantial reduction in searches a lot of directories are required.

2) Coding - The directory will often contain its information coded in a form much more convenient for searching than the data of the file. This is particularly true in a library, where the relationship between a book and an attribute may be spread across several chapters of text, but can be defined in a few words on a catalog card. This effect is what makes the redundancy of the previous point effective. Further levels of redundancy would be achieved by finding additional codings of the same information.

3) Speed of Access - Since the directory is smaller, in general, it can be stored in a storage device which has faster access times than that of the main file. For example, a card catalog is more easily searched than a shelf of books, and core memories are much faster than the disk files they hold directories for. This effect is small, averaged over all questions, since the amount of fast-access information is small compared to the size of the file. The exception to that is when file access is effectively

limited only to those questions which can use the directory information, in which case the speed difference is of course significant.

4) File Position Pointers - When a file is ordered on usable information, a system of pointers is necessary to guide access to various entry points in the ordering sequence. For example, this function is served by 1) the range indicators on the front of each tray of a card catalog, 2) column headings on a page of a dictionary, or 3) a hash code algorithm of a computer file. This pointer information is quite small compared to the size of the file, but serves a vital role in utilizing the ordering information of the file. This pointer data is distinct from the other uses of a directory, and requires a different analysis.

Perhaps a little theory will explain this position information a little better.* If s items are put in order, there are s! ways of ordering them, so that one ordering represents $\log_2 s!$ bits of information. By Stirlings approximation:

$$\log_2 s! \simeq s \log s - s \log_2 e + \frac{1}{2} \log_2 2\pi s$$
$$\simeq s(\log s - 1.44).$$

---

* This approach was developed by Prof. P. Elias, who is preparing a paper showing further results in this area.

Thus a set of r-bit items can be represented, when ordered, by about $r-\log_2 s + 1.44$ bits. This theoretical minimum can actually be approached by difference coding, whereby the items are ordered by increasing value, and each item is coded as the difference in its value over the previous item. While this coding is compact, search lengths are quite bad because finding an item in the middle of the file requires adding up all the differences from the beginning. As slight redundancy is added, search lengths can be reduced, as in this scheme:

1) Order the items by increasing value, and store only the $r-\log_2 s$ least significant bits.
2) Create a directory which gives the value of the high-order $\log_2 s$ bits for each item by using a unary difference code: If the log s bits of item b_n have value x greater than that of b_{n-1}, this is coded as x 1's and a 0; the zero being a marker to separate codes. This costs at most 2s bits in the directory: s 1's and s 0's.
3) Searching requires an exhaustive search of the directory (2s bits), and a single access to the main file (occasionally more than one) for each configuration present and accessed.

This scheme requires 2 bits per item above the r-log s bits, rather than the 1.44 bit theoretical minimum. However, it gives fairly short searches which begin to approach the theoretical minimum. Further redundancy yields shorter searches yet, but not by such big margins.

In short, a little position information in a directory is necessary to utilize the ordering of the main file, but the amount

of directory information is quite small for this function. Beyond that, additional information in the directory yields only slight help in total search length for general cases.

5.3 OTHER FILE ORGANIZATION TECHNIQUES

The preceding discussions failed to mention a number of standard filing techniques. This is because of the previous assumptions that the file is static (or has many more references than updates) and that question sets were of the partial match type. Alternatives are discussed here.

Hash Coding. In some systems, a mathematical algorithm is used to calculate a file address from the information supplied by the user. This is used when:

a) The user supplies the full key describing the data, rather than a partial specification, and

b) Not all keys are equally likely, but rather there are "hot spots" in the key field such that some key areas are more densely used (for example, in zipcodes the city areas get more mail of some types than country areas); this still assumes a large selection of keys for the items used, $2^r >> s$.

The normal file method when b) is not a problem (keys are equally likely), is to:

1) Order the file by key values,

2) When searching, estimate from the specified key where in the file it will appear, by its value relative to the largest and smallest key,

3) After the first access, compare the specified key with the retrieved key, create a new estimate of item location by interpolation, and repeat this step until the item is found.

When the key values are unevenly used, however, the above procedure takes longer than it does on random keys. To offset this, the hash code procedure is used to map the original keys into a new set which are evenly used (i.e. the hashing algorithm is selected to break up the correlations in the key field).

This procedure always takes adjacent key values and distributes them randomly over the new key field, for that is how correlations are broken up. This has real troubles with incompletely specified questions, however, because searches for such questions depend on adjacencies of similar keys to reduce search lengths. That is, the hash code effect of scattering items with similar keys is exactly the wrong thing to do when dealing with partially specified keys. Hash codes therefore appear in only a limited range of computer file problems, and are not really applicable to large files.

An alternative (and sometimes augmentation) to hash coding is the use of redundancy in memory. If memory has more slots than items, the items can be distributed according to a linear distribution of keys rather than the uneven real distribution. The more redundancy available, the more even the distribution will be, and the faster the searches. This problem has not been fully analyzed to date.

List Structures. One system which is frequently considered for files having several access paths to each item is list structuring. Here each item having a particular attribute also contains a pointer to another item containing that attribute. Access to the attribute then consists of following a chain of these pointers from item to item through the file. Frequently items are ordered in the chain by some other attribute value.

The list system can be seen to be nothing more than a directory system where the directory is scattered throughout the file. That is, the pointers of the list system are the same as directory entries, only stored differently. The difference between the systems lies in memory allocation in rapidly changing files.

List structures store their pointers along with the data, so as the file shrinks or expands, in total or in parts, the memory allocation of pointers is handled automatically with allocation for the data. Further, the algorithm for accessing the pointers does not change as their quantity changes. If the pointers were segregated into a separate portion of memory, that portion would have to be expanded and contracted to meet changing file needs, and would have to be continually restructured as various portions of the file become larger ("hot spots" in the file).

The directory system has the advantage that with segregated pointers, the pointers can be searched without having to access the items of the file. This is an advantage in all systems where the file is in a storage facility which is not fully random access (and very few large files can be put in a true random access facility).

The directory is especially superior if it can be stored in a faster access mechanism.

This is the typical trade-off between speed of up-date and speed of access. For files where new data is added more frequently than old data is accessed, list structures offer very easy addition of new items at the cost of slower access during retrieval. For files with more frequent accesses, directories permit location of items with fewer memory accesses, but require more effort to up-date.

Note that the search-length analysis for list systems is the same as for a directory system, with equivalent trade-offs between recall and relevance, etc.

Inverted Files. A common problem in file storage is whether to store data:

a) Item-oriented, where each item is coded explicitly with a list of its related attribute values; the items being ordered according to some attribute values, so that some of the attribute information can be omitted and replaced by the item address.

b) Attribute-oriented (inverted), where each attribute is coded explicitly with a list of related items; a multi-attribute search requires a logical intersection of the item lists under the specified attributes; the attributes can be arranged in order so that some item occurrences can be predicted by attribute address, and these item identifiers can be omitted from the attribute lists.

In storage costs, these two systems are theoretically identical, but practically it is usually better to organize on the set having the greatest number of elements. That is, for r attributes and s items, the inverted file is better if r>s, for two reasons of coding efficiency:

1) For an attribute-item occurrence of density ρ, each occurrence should cost log ρ bits in theory, but that requires difference coding for the normal case of small ρ. For convenient retrieval the full coding is used, so the cost is log r bits in an item file and log s bits in an inverted file. Thus if the file is organized on the larger of r or s, then the inefficiency is minimized.

2) Theoretical minimum storage requires ordering on both dimensions: Attributes within an item-file can be ordered so that the first $\frac{\log r}{\rho \log \rho}$ items have predictable attribute patterns, and need not be stored explicitly; likewise in an inverted file item ordering can reduce the number of attributes actually stored. This saving is seldom achieved because it is small relative to the additional complexity of search algorithm which results. The loss of efficiency in not ordering is minimized if the file is organized on the larger of r or s. This factor does not apply if the file is not ordered at all, even in its main dimension of organization, for in that case both organizations are equally inefficient.

In search length costs, the size of the user question is important. If the user tends to specify numerous attributes yielding a small response, the item file is best. Conversely, a few attributes and a long consequent response are best handled by the inverted file. In-between cases are dependent on the physical configuration of equipment available. Inverted files require substantial short-term memory to hold item lists during the process of intersection of the lists from several attributes. Further, inverted files do not have a complete description of an item anywhere, so they are at a strong disadvantage where users require the full item description as output. Item files, however, tend to require searching a greater percentage of the file bits, especially when the number of attributes is large. Inverted files give approximately the same search lengths for all questions, while item files will give a large variance in search lengths, depending on the number of specified attributes which fall into the set on which the items are ordered.

In short, the trade-off between item files and inverted files is decided on factors local to each implementation. It is interesting to note, however, that all analysis to date indicates that any in-between organization (organized partly on items and partly on attributes) is always less efficient than one or the other extreme case.

5.4 AREAS OF FURTHER RESEARCH

There are a number of very interesting topics in file organization which deserve further research. They have been mentioned at

various points in this document, but deserve more specialized attention. These are problems for which solutions are quite possible, I believe, if the right theory is found.

1) Find a tight lower bound on search length for partial match questions, regardless of file organization.
2) Find a decent bound or estimate for search length in redundant files, for large redundancies (say a factor of s).
3) Develop a theory which predicts an "optimum" user question ensemble based on file statistics (especially inter-attribute correlations); this involves figuring out what the user objectives are in asking questions of the file, and assumes that users adopt their questions to fit the information available.
4) Find a means to better analyze the problems of file access with fully-specified keys, namely: A) the effects of small redundancies on search length, and B) the trade-offs between retrieval speed and update speed.
5) Find methods to extend the $p_n \log p_n$ analysis to give more detailed answers for partial-recall questions, such as a search length bound for questions with all recall levels equally likely.

I will be happy to hear from or talk with anyone with interest or results in these areas.

APPENDIX

A COMBINATORIAL ANALYSIS OF THE BUCKETING PROBLEM

A partitioning problem which appears in several forms in computer-related topics is discussed here, in its guise as a file organization problem. It is the question of how to allocate file information into memory "buckets" so as to minimize the average number of buckets which must be searched to answer user requests. The approach used is a combinatorial one, in essence counting those attribute combinations which occur most frequently in the user requests. One result is a rigorous lower bound on search lengths for partial-match and near-match types of requests, within certain file organizations and with simplified file statistics. A second result is a measure of improvement of search length as file storage redundancy is introduced (items repeated in the file).

A.1 THE BUCKETING PROBLEM

Given: An r-dimension binary space, called "key-space".

A set of s items, which appear as marked points in key-space; $s << 2^r$.

An ensemble of user questions, each question being a sub-volume of key-space.

A set of B buckets, each of which is to "cover" a fixed size sub-volume of key-space.

Objective: Partition key-space (i.e. divide key-space points into bucket assignments) so as to minimize the average number of buckets required to cover user questions (average search length).

Specifically, the file problem consists of s items in a file, each of them having a key which is one of the 2^r binary keys in key-space. When a user asks a question, (e.g.) "Which file items have keys whose first three bits have value one?", he is specifying a sub-space of key-space which is to be searched. The file response to this question is a list of all marked points (file items) which appear within the user-selected sub-space. Each file bucket contains some subject category or other sub-set of the key-space, and the objective of file organization is to select bucket sub-sets which minimize search length.

A key characteristic of the analysis here is that the key information describing each file item is much greater than sufficient to just distinguish the s items. That is, $r >> \log_2 s$. The user may identify a single item through many different descriptions, each one giving only part of the key information. This flexibility of access for the user is what makes file organization difficult, and files where $r = \log_2 s$ are not interesting. Of course bucketing is only one of several ways to organize such a file, but meaningful answers derived for this special case will give needed insight into file storage in general.

To further simplify the mathematics of this analysis, the following assumptions are made:

1) The s marked points occur independently and equally likely throughout key-space, or equivalently, all key-space points are equally likely to appear in the file (thus each bucket can be assigned an equal chunk of key-space, without need to compensate for correlations).

2) The uneven distribution of marked points in buckets can be tolerated (proper use of a directory can compensate for such unevenness). While each of the B buckets will have the same expected number of s/B items in it, the actual number will vary due to the random process of selecting file points. We assume this causes no trouble, but the resulting analysis is therefore not fully descriptive for files with tight memory packing or high update rates.

User Question Statistics. For the sake of discussion here, two forms of user question ensemble are studied. These are unrealistically simplified, of course, but yield good insight into the more complicated real situations.

Partial Match Requests: The user specifies a 1 or 0 value for any t of the r attributes, and he wants items which exactly fit all those t values. The $2^t\binom{r}{t}$ such questions are equally likely to be asked. Each question specifies 2^{r-t} points in key-space, which will contain on average $s2^{-t}$ file items.

Near Match Requests: The user specifies a full r-bit key, giving a one or zero for each bit, and he wants those items which fall within Hamming distance d of the specified point in key space (i.e. all points which come near the specified point). The 2^r such questions are equally likely. Each question specifies $\sum_{x=0}^{d}\binom{r}{x}$ keys, containing $\sum_{x=0}^{d}\binom{r}{x}s2^{-r}$ file items on average.

Example A-1: A File for Partial-Match Requests

The user specifies t of the r attribute bits, and he wants all items falling within the intersection of those t attribute values. Presume r = 15, s = 2^{11}, B = 2^{11}, and t = 11. Thus the 2^{11} buckets will contain on average one item each. Presume that each bucket will cover those points having the same first 11 bit values. For example, the 16 keys described by 10101010101ØØØØ will all be assigned to one bucket. This is the conventional ordered-file method of organization.

When a question is received, its first 11 bits are used to compute bucket addresses, with each Ø replaced alternately by 1 and 0. For example, the question ØØØ01010101Ø111 specifies eight of the first eleven bits with three missing, so that 2^3 = 8 buckets must be accessed to examine all points covered by the question.

Of the $2^{11}\binom{15}{11}$ different user questions:

24% have 4 of the first 11 bits missing, requiring 16 accesses
48% " 3 " " " " " " " 8 "
24% " 2 " " " " " " " 4 "
4% " 1 " " " " " " " 2 "
.07% " 0 " " " " " " " 1 "

Thus, searching in this example requires 8.77 bucket accesses on average.

Strategy of Analysis. The problem parameter of interest, average search length, is very difficult to compute directly for most cases of file organizations and question ensembles. The first step of analysis is the derivation of a search length bound which is simplier to compute. Some examples are given to indicate the tightness of the bound. Then this bound is used to examine the effect on search length of 1) file size, 2) type of question asked, 3) type of organization used, and 4) use of redundant file implementations.

As a goal of the analysis, three particular results are primary:

1) Proof that average search length must continually increase as key-space size (r) is increased, regardless of file organization used.
2) A demonstration that file storage redundancy (repetition of items at various points in the file) improves search lengths, but not rapidly.
3) An indication that partial-match and near-match questions offer the same order-of-magnitude difficulty in access.

A.2 SEARCH LENGTH BOUND

Definition: T_Q is the number of questions in Q. Thus Q is composed of questions q_i, $1 \leq i \leq T_Q$, all equally likely.

Definition: P is a partition on keyspace, being a particular assignment of key points to buckets (a file organization).

Definition: L is a search length for a P and Q, namely the average number of buckets accessed for questions in Q.

where the inequality on average reciprocal search length follows from $\overline{(1/x)} \geq 1/\bar{x}$, with equality when all ℓ_i are the same.

Part 2.

To define the function W(Q,P), an intermediate definition is useful:

Definition: A Question-Point-Pair (QPP) is a counting unit which occurs whenever two points in a bucket occur together in the same question.

Definition: $W_b(Q,P)$ is the number of QPP's in a bucket b of P under the question ensemble Q.

That is, W(Q,P) gives a count of the number of times a pair of points which share a bucket also share a question. A pair of points which fall together under m questions produce a value of m QPP's in their bucket total. A question which covers n_j points in a bucket will produce $\binom{n_j}{2}$ QPP's.

Consider now a specific bucket, b, and all those questions which access it. Each such q_j covers n_j of the points in the bucket. It is not important how many such questions there are, so we do not bother to assign bounds to j. N_b will be defined as the average of useful points accessed for this bucket:

$$\text{average } (n_j) = N_b \ , \tag{A.4}$$

or equivalently,

$$\sum_j n_j = \sum_j N_b \ . \tag{A.5}$$

To calculate W(Q,P), first re-express n_j in terms of its deviation from average:

$$n_j = N_b + x_j \qquad \sum_j x_j = 0. \tag{A.6}$$

Merger of Parts 1 and 2.

Since N is the average of useful points in all bucket accesses, there must be at least one bucket whose average N_b is as large as N. Therefore there must be at least one bucket in the file such that:

$$V(Q)/L \leq N_b \quad . \tag{A.13}$$

For the files of interest in this paper, W(Q,P) is the same for all buckets, so this proves the theorem. QED

A.3 EXAMPLES OF BOUND CALCULATIONS

Example A-1 continued:

For the partial-match example described above, we count QPP's as follows: Two points which are distance-one apart may co-occur in $\binom{r-1}{t}$ different questions, since the t specified attribute values may appear in any of the r-1 attribute values the points have in common. Of the $\binom{16}{2}$=120 point pairings in the bucket assignment of the example, 32 are distance-one pairings. Extending this to higher distances:

pair distance	number of such pairs per bucket	×	questions in common	=	QPP's
1	32		$\binom{14}{11}$ = 364		11,648
2	48		$\binom{13}{11}$ = 78		3,744
3	32		$\binom{12}{11}$ = 12		384
4	8		$\binom{11}{11}$ = 1		8
	120				15,764 =W(Q,P)

For the partial-match questions we have already found these values:

$V(Q)$ = points per question = $2^t = 2^4 = 16$

T_Q = number of questions = $2^{11}\binom{15}{11} = 2^{11} \cdot 1365$

$B = 2^{11}$.

Then using the theorem (A.1):

$$2^4/L \leq + 1 + \frac{2 \cdot 15{,}764}{16 \cdot 1365} = 2.44$$

$L \geq 16/2.44 = 6.5$ bucket accesses.

This compares with the actual value of L = 8.77 accesses calculated previously. The difference comes because the various questions in Q access the buckets with unequal efficiency.

Example A-3: A Different File Organization

For comparison, suppose the file organization of example two were used with the file problem of example one. That is, points will be assigned to buckets in d=1 spheres, for use with partial-match questions. The QPP's of this organization are:

pair distance	number of pairs	$\binom{r-d}{t}$	QPP's
1	r = 15	364	5,460
2	$\binom{r}{2}$ = 105	78	8,190
		120	13,650

For this case, $N \leq 1.25 + 1 = 2.25$ points per access, so that

$L \geq 7.1$ bucket accesses.

This implies that the sphere-packing of example two is an inferior organization for the partial-match question. Direct search length calculations for this case are a bit tricky, but can be carried out to show a real value of L = 12.

Example A-3 shows the principal weakness of the estimator. It was derived assuming an equal distribution of accessed-points over bucket accesses, namely that all n_i are equal. In Example A-3, due to the peculiar geometry of the organization, each bucket access yields one point or five points, nothing in-between. This produces a wide variance around the mean of two, so the bound is not very close to the real value of L.

Large r Examples

Figure A-1 shows the conditions of Example A-1 extended over a range of r values, giving a comparison of real and estimated search lengths. Figure A-2 shows the same information for a larger file system, with one million buckets.

These figures indicate that the bound tends to understate search length by a factor of 2 to 3 for large r. Calculations for other examples confirm this as a typical error, although the error may get notably larger for poor organizations (such as in Example A-3).

A.4 COMPARISON OF FILE SYSTEMS

One application of the search length bound is to compare the relative difficulties in accessing files of various types. How do near-match questions compare to partial-match questions in access difficulty? What is the best geometry of organization for each question type?

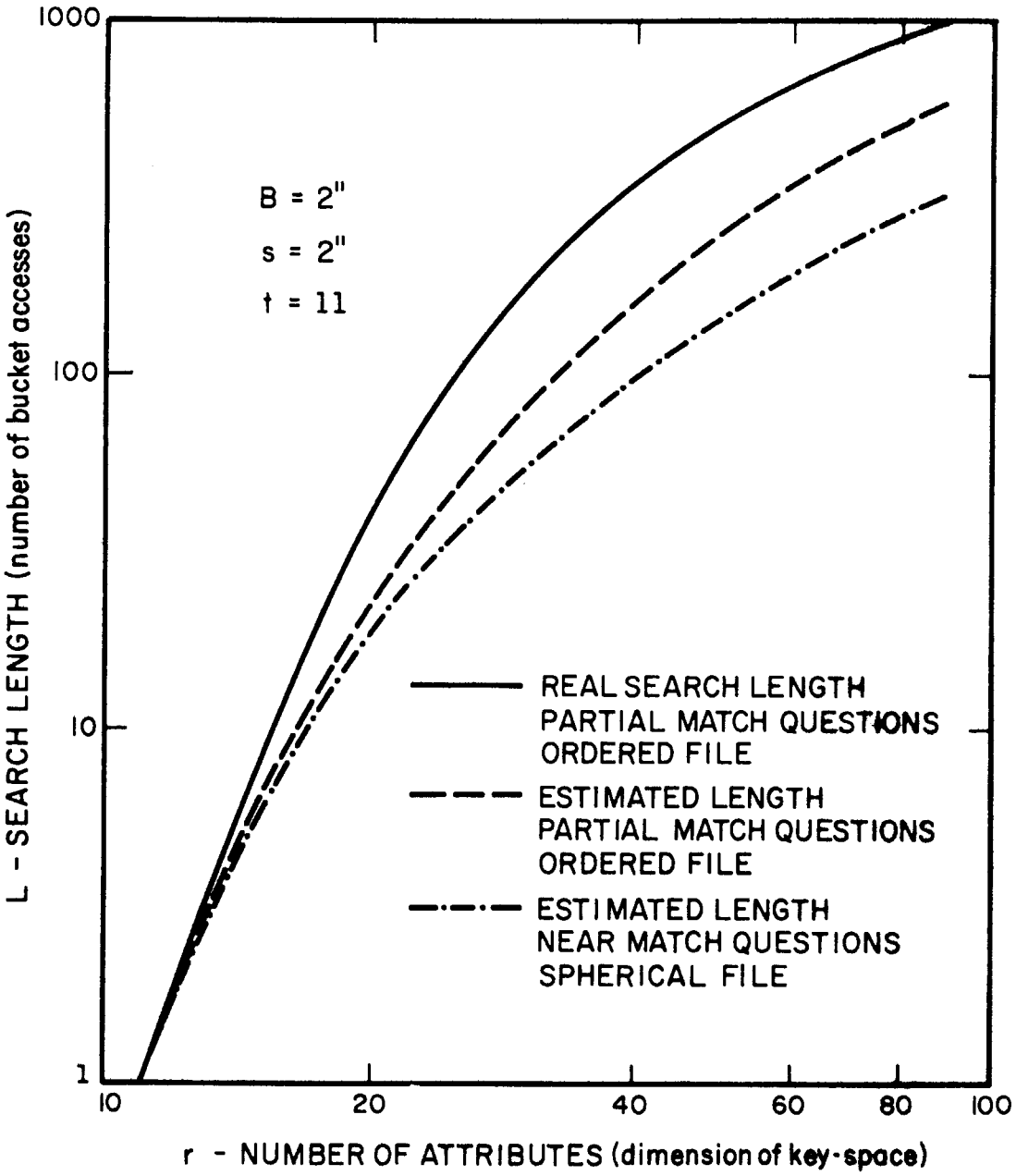

FIG. A.1 SEARCH LENGTHS ON 2000 ITEMS

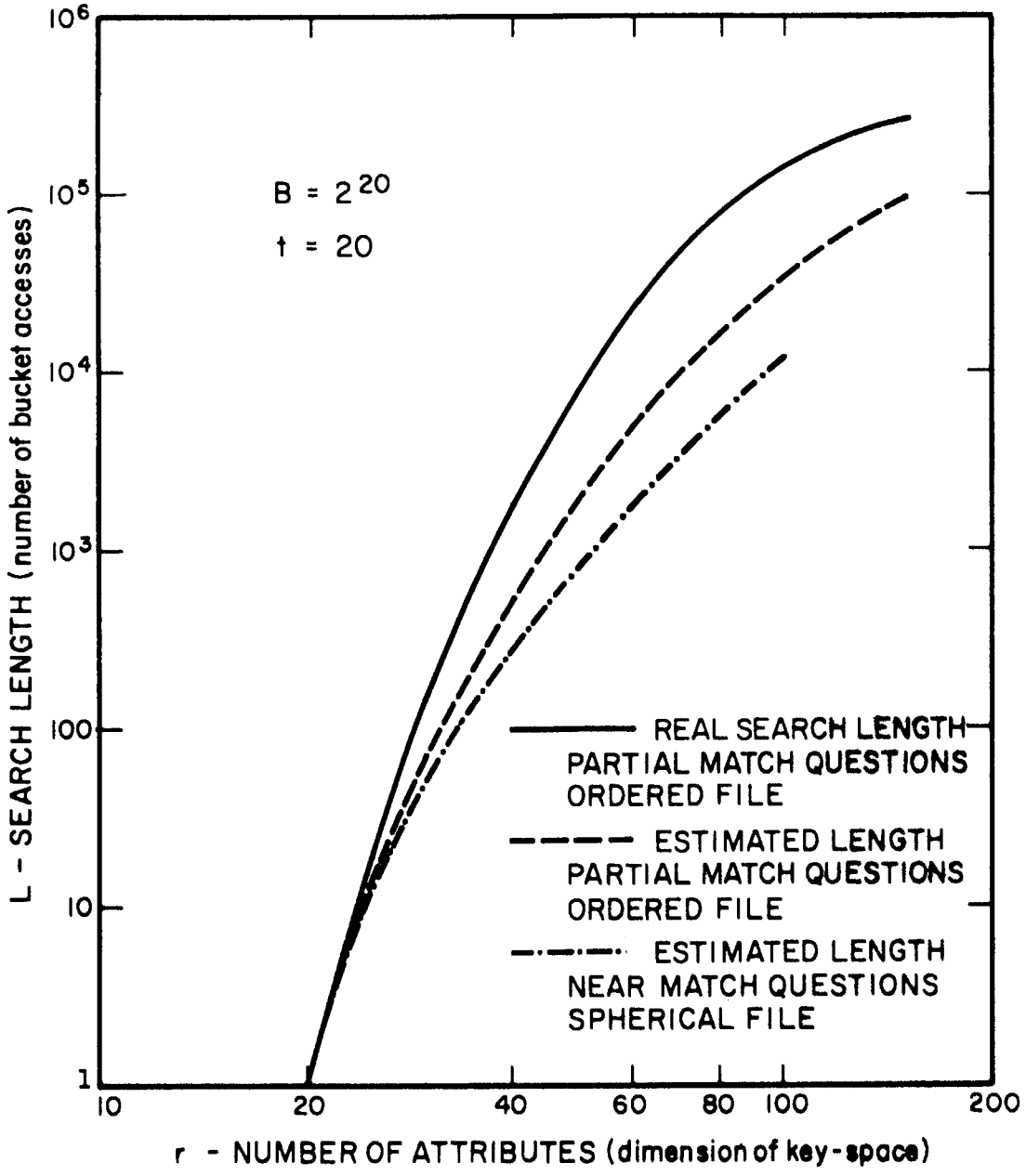

FIG. A-2 SEARCH LENGTHS ON 10^6 ITEMS

Figures 1 and 2 compare partial-match and near-match question types, with each using the file organization matching its geometry. For these curves, the number of buckets and questions is held constant as key-space size is increased. For the near-match questions, this means that the Hamming distance d of the question must increase. For example, at $B = 2^{20}$ and $r = 100$, each question covers all points within distance 26 of the specified center point. Actually, these sphere distances are not integers, and approximations must be used to handle the fractional parts.

The clear implication of these curves is that near-match questions are more easily handled for large key-spaces, but only by a factor of two or three at most. Intuitively, this is justified by the observation that some partial-match questions require an exhaustive search of the file (i.e. access all buckets), while that does not occur with the near-match questions. This advantage for near-match questions must be balanced against two other factors:

User Flexibility: The number of questions that a user can ask which cover a given point in key-space indicates his flexibility in specifying a file item. When all questions cover 2^{r-t} points, this is proportional to T_Q.

Partial Match: $T_{Qp} = \binom{r}{t} 2^t$

Near Match: $T_{Qn} = 2^r$

For small r, with fixed t, $T_{Qp} > T_{Qn}$, and significantly so in many cases. However, the values are equal around $r = 4t$, and near-match questions offer the greater flexibility for larger r.

User Information: The amount of information a user must specify in posing a question is:

Partial Match: t bits

Near Match: r bits.

As r gets large, this becomes a very significant difference, which will rule out near-match questions for most applications.

In the balance, it appears that partial-match questions have the advantage for all values of r, but the advantage is small in many cases. The outstanding exception to this rule would be an application where r>>4t and the cost of user-supplied information can be kept low without reducing flexibility. I cannot think of an example of such an application.

When considering the question of what bucketing system to use with a particular question type, the bounding procedure permits two observations to be made:

1. The QPP calculations weight low-distance point-pairs much more heavily than high-distance pairs; therefore, bucket assignments should emphasize the preservation of low-distance pairings within buckets. The two organization systems discussed above appear to do this better than any other possible partitioning.
2. Equality in the bound is best approached when each bucket accessed tends to yield the same number of useful points (all n_i are equal). This condition appears to be very difficult to approach in practice, but perhaps is a good thinking tool.

Partial-Match Questions. A simple analysis, similar to the preceding near-match one, assumes a bucket geometry whose pair distances are one-half distance-1 and one-half distance-2 (the best ratio possible). The details are omitted here, but the result is:

$$L \geq \frac{B2^{-t}}{\frac{1}{2}(1 - \frac{t}{r})(2 - \frac{t}{r-1}) + 2^{-r}B} \qquad (A.15)$$

For the values of Example 1, r = 15, t = 11, this says that

$L \geq 4.44$ accesses regardless of file organization.

This bound could be improved somewhat with more complicated arguments, but it remains weak for larger values of r (when $2^r/B >> r$).

For a large bucket analysis, assume that a bucket is optimumly packed with respect to every point it holds. That is, for a given point, all of its d = 1, 2, 3, ... x neighbors are in the bucket, up to the point that bucket capacity is reached. This assumes that the bucket holds points for a sphere of radius x, where x is fixed by:

$$\sum_{d=0}^{x} \binom{r}{d} = 2^r/B, \qquad \text{since the point has } \binom{r}{d} \text{ distance-d neighbors.}$$

This optimum packing gives an upper bound on the number of QPP's possible in a bucket:

$$\text{QPP's} \leq 2^{r-1} B^{-1} \sum_{d=1}^{x} \binom{r}{d}\binom{r-d}{t}$$

$$= 2^{r-1} B^{-1} \sum_{d=1}^{x} \binom{r}{t}\binom{r-t}{d} . \qquad (A.16)$$

Then, substituting A.16 into A.1:

$$\frac{V(Q)}{L} - 1 \leq \frac{2^{-r+1} \cdot B \cdot 2^{r-1} \; B^{-1} \sum_{d=1}^{x} \binom{r}{t}\binom{r-t}{d}}{\binom{r}{t}} = \sum_{d=1}^{x} \binom{r-t}{d} \qquad (A.17)$$

$$\text{or,} \quad \frac{V(Q)}{L} \leq \sum_{d=0}^{x} \binom{r-t}{d} \quad \text{where x is determined by} \quad \sum_{d=0}^{x} \binom{r}{d} = 2^r/B. \qquad (A.18)$$

Solving this relation requires several tedious steps which will be omitted here. Using the Vandermonde Convolution Formula and numerous variable changes, it was determined that:

$$\sum_{d=0}^{x} \binom{r-t}{d} \leq \sum_{d=0}^{x} \binom{r}{d} 2^{-t} + \binom{r-t}{x-t/4} \cdot \frac{t}{2} \qquad (A.19)$$

Using A.19, and Stirling's Bound:

$$\binom{z}{y} \leq \binom{z}{z/2} \leq 2^z \sqrt{\frac{2}{\pi z}} \; e^{1/12z}$$

$$\frac{V(Q)}{L} \leq 2^{r-t}/B + \binom{r-t}{x-t/4} \cdot \frac{t}{2} \leq 2^{r-t}/B + \frac{t}{2} 2^{r-t} \sqrt{\frac{2}{\pi(r-t)}}$$

$$\frac{2^{r-t}}{L} \leq 2^{r-t}\left(B^{-1} + \sqrt{\frac{t^2}{2\pi(r-t)}}\right)$$

$$L \geq \frac{1}{B^{-1} + \sqrt{\frac{t^2}{2\pi(r-t)}}} \qquad (A.20)$$

As r becomes very large, the radical term goes to zero, and L approaches B. That is, as r becomes big with t fixed, file access approaches an exhaustive search no matter how the file is organized.

A.6 REDUNDANCY

The above analysis assumed that every file space point was covered by only one bucket, so that any question referring to that point had to access that bucket. Now we permit a point to appear some number of times, M, in different buckets. For each question, then, there is a choice of M buckets in which to inspect the point. In this system the file implementation takes M times as much memory as the minimum configuration. How does this redundancy help to reduce search time?

The number of questions accessing a given point remains constant, but they are now divided over M occurrences of the point. Equivalently, the number of buckets B in the system is increased by M, but the size of each bucket is unchanged so the W(Q,P) values remain the same. Evaluating the basic bound, A.1, then shows that L decreases proportionally with M:

$$L(M) = C \frac{L(1)}{C'M+1}$$

where C and C' are constants, and L(1) is the non-redundant search length.

The constant C' is virtually always larger than one, so L(M) is just inversely proportional to M.

There is one flaw in this analysis, due to the error in the search length estimator. If the non-redundant system has long search lengths due to uneven distribution of points in buckets (Example A.3), redundancy can permit a more even distribution and consequent reduction in inefficiency. That is, the real search length will come much closer to the lower bound in a redundant system. The extent of this effect is not known, but is probably limited to small M. It generally can produce at most a factor of 2 or 3 improvement in L, for that was the degree of error in the original bound.

Actual search times in a redundant system are very difficult to compute. Two examples of limited range are given here:

Assume Partial-Match questions and ordered file organizations, with M copies of the file organized on disjoint sets of attributes.

Case 1. $r = 100$ $B = 2^{20}$ $t = 20$

M	Theoretical Bound	Real Search Length
1	$L \geq 33,557$	$L = 115,381$ accesses
2	$L \geq 16,778$	$L = 48,211$
3	$L \geq 11,186$	$L = 30,427$
4	$L \geq 8,195$	$L = 22,496$

Case 2. $r = 66$ $B = 2^{11}$ $t = 11$

1	$L \geq 408$	$L = 758$
2	$L \geq 204$	$L = 445$
3	$L \geq 136$	$L = 329$
4	$L \geq 102$	$L = 269$
5	$L \geq 82$	$L = 230$

This analysis of the effects of item repetition is the only treatment I have found which is useful on the redundancy question. Though flawed by some looseness for small M, it is a rigorous lower bound which cannot be surpassed. It provides a firm conclusion that storage redundancy cannot give remarkable decreases in search length.

A.7 CONCLUSIONS AND CONJECTURE

The above derivation has provided a rigorous lower bound on file search length. While the bound is not particularly tight, it is much easier to calculate than real search lengths in some organizations. Further, it becomes possible to calculate search length bounds irrespective of file organization used, and to extend the results to large key-spaces.

The conclusions reached for the model used were:

1) File search length grows with increased flexibility available to the user; as the number of attributes available to the user grows larger, the search length approaches exhaustive search, regardless of what file organization technique is employed.
2) Redundancy in file implementation can only linerly decrease search lengths, with some exceptions for small redundancies; if each file item appears M times in the implementation, this reduces search length only to 1/M its non-redundant value.
3) Partial-match and near-match question types fall into the same range of difficulty for file access; this implies that

other question types (such as Boolean sums of products of attributes) will tend to cause equivalent search lengths.

For mathematical convenience we have taken a file problem with specialized statistical properties as the subject of this discussion. Real library problems are of course different in these aspects: 1) Not all questions will be equally likely (e.g. real users tend to specify more 1's than 0's as attribute values), 2) not all file keys are equally likely (attributes will be correlated), and 3) only a limited range of file organization techniques was allowed here (e.g. directory techniques were not discussed).

Despite these limitations on the model used, I feel that the results obtained are generally indicative of what would be found in the more complicated cases if we could analyze them.

BIBLIOGRAPHY

General

1. J.C.R. Licklider
Libraries of the Future
MIT Press, 1965.

2. Y. Bar-Hillel
Language and Information, 1964
See especially Chapter 13, "The Future of Machine Translation".

3. G. Salton
Automatic Information Organization and Retrieval
McGraw-Hill, 1968.

4. M. Minsky and S. Papert
Perceptions, An Introduction to Computational Geometry
MIT Press, 1969. See especially Chapter 12.

Interesting Work of Narrower Interest

5. C. Cleverdon, J. Mills, and M. Keen
ASLIB - Cranfield Research Project (2 vols.)
Cranfield, England, 1966.

6. J.A. Swets
"Effectiveness of Information Retrieval Methods"
American Documentation, January 1969, p. 72-89.

7. M.M. Kessler
"Comparison of the Results of Bibliographic Coupling and Analytic Subject Indexing"
American Documentation, Vol. 16, No. 3, July 1965, p. 223-233.

8. F. Jonker
Indexing Theory, Indexing Methods, and Search Devices
Scarecrow Press, 1964.

9. S.P. Ghosh and C.T. Abraham
"Application of Finite Geometry in File Organization for Records with Multiple-Valued Attributes"
IBM Journal of Research and Development, March 1968, p. 180-187.

Statistical Association Studies

10. Statistical Association Methods for Mechanized Documentation
1964 Symposium Proceedings, Edited by Stevens, Giuliano, and Heilprin; National Bureau of Standards Miscellaneous Publication 269. It contains:

11. F.B. Baker, p. 149-155
"Latent Class Analysis as an Associative Model for Information Retrieval"

12. J.H. Williams, p. 217-224
"Results of Classifying Documents with Multiple Discriminant Functions"

13. H. Borko, p. 245-251
"Studies on the Reliability and Validity of Factor-Analytically Derived Classification Categories"

14. J.B. Lovins
"Error Evaluation for Stemming Algorithms as Clustering Alrogithms"
Journal of the American Society for Information Science, Vol. 22, No. 1, Jan. 1971, pp. 28-40.

15. G. Salton
Information Storage and Retrieval
NSF Report No. ISR-16, Cornell University, Sept. 1969. It contains:

16. J.W. McNeil and C.S. Wetherell
"Bibliographic Data as an Aid to Document Retrieval"

17. E.L. Ivie
"Search Procedures Based on Measures of Relatedness Between Documents"
MIT Doctoral Thesis, June 1966, Project MAC TR-29.

18. K.S. Jones and D. Jackson
"Current Approaches to Classification and Clump-Finding at the Cambridge Language Research Unit"
Computer Journal, May 1967, Vol. 10, No. 1, p. 29-37.

Mathematical Tools

19. R. Gallager
Information Theory and Reliable Communication
Wiley, 1968.

CS-TR Scanning Project
Document Control Form

Date : 1/23/96

Report # LCS-TR-88

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 164 (170-IMAGES)
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:

☒ DOD Form ☐ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:
IMAGE MAP: (1-164) UNNUMBERED TITLE PAGE, 2-164
(165-170) SCANCONTROL, COVER, DOD, TARGETS(3)

Scanning Agent Signoff:
Date Received: 1/23/96 Date Scanned: 1/24/96 Date Returned: 1/25/96

Scanning Agent Signature: Michael W. Cook

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Massachusetts Institute of Technology Project MAC	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP None

3. REPORT TITLE

Bounds on Information Retrieval Efficiency in Static File Structures

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Ph.D. Thesis, Dept. of Electrical Engineering, May 1971

5. AUTHOR(S) *(Last name, first name, initial)*

Welch, Terry A.

6. REPORT DATE June 1971	7a. TOTAL NO. OF PAGES 166	7b. NO. OF REFS 19
8a. CONTRACT OR GRANT NO. Nonr-4102(01) b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) MAC TR-88 (THESIS) 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES None	12. SPONSORING MILITARY ACTIVITY Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301

13. ABSTRACT

This research addresses the problem of file organization for efficient information retrieval when each file item may be accessed through any one of a large number of identification keys. The emphasis is on library problems, namely large, low-update, directory-oriented files, but other types of files are discussed. . .

The principal analysis variable is item relevance, the probability that a file item accessed is actually useful, which is a measure of retrieval efficiency. An upper bound on average relevance is derived, and is found to give useful results in two areas. First, it shows that retrieval efficiency is determined primarily by catalog size (amount of information stored) and user question statistics, with only second-order effects due to type of catalog data and file structure used. Second, it is used to evaluate various indexing procedures proposed for libraries and to suggest improved experimental procedures in this field.

14. KEY WORDS

Information Theory
File Organization
Information Retrieval
Classification
Directories
Relevance
Libraries

DD FORM 1 NOV 65 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**