21964

NAME OF AUTHOR/*NOM DE L'AUTEUR* SCHUEGRAF ERNST J.

TITLE OF THESIS/*TITRE DE LA THÈSE* THE USE OF EQUIFREQUENT FRAGMENTS IN RETROSPECTIVE RETRIEVAL SYSTEMS

UNIVERSITY/*UNIVERSITÉ* UNIVERSITY OF ALBERTA EDMONTON

DEGREE FOR WHICH THESIS WAS PRESENTED/ *GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* Ph. D

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE DEGRÉ* 1974

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* H. S. HEAPS

DATED/*DATÉ* 11 June 74 SIGNED/*SIGNÉ* [illegible]

PERMANENT ADDRESS/*RÉSIDENCE FIXÉ* P.O. Box 55 ST. F.X.U.
ANTIGONISH, N.S.

NL-91 (11-73)

THE UNIVERSITY OF ALBERTA

THE USE OF EQUIFREQUENT FRAGMENTS IN RETROSPECTIVE RETRIEVAL SYSTEMS

by

© ERNST J. SCHUEGRAF

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1974

THE UNVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled " THE USE OF EQUIFREQUENT FRAGMENTS IN RETROSPECTIVE RETRIEVAL SYSTEMS " submitted by Ernst J. Schuegraf in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

[illegible]
Supervisor

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]
External Examiner

Date May 27, 1974

ABSTRACT

A new approach to organization of large retrospective document retrieval systems is proposed, based on equifrequent fragments for indexing and compression elements. The critical parameters of such a system are isolated and are used to design an algorithm for the selection of dictionary fragments which optimize these parameters. A datastructure suitable for implementation of the algorithm is described and experimental results for two sample databases are presented. System parameters are found to be simple functions of a threshold frequency, the only input parameter of the selection algorithm. The effect of this threshold, and changes necessitated in the system organization are studied in detail.

It is shown that finding the minimum storage form of a database compressed with equifrequent fragments is equivalent to solving a shortest path problem. Three compression algorithms are investigated and their performance tested on a sample database. Furthermore an algorithm for query processing in a fragment oriented retrieval system capable of handling truncated search terms is described, its retrieval performance tested and comparative data for the sample is given. Two necessary conditions for successful operation of such a system are postulated.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Prof. H.S. Heaps, my supervisor and spiritual mentor, for his advice, quidance and criticism throughout the course of the research and the preparation of this thesis. I am also indebted to Dr. I.N. Chen for his assistance and support; as well as to Prof. D.M. Heaps for providing the needed encouragement. Thanks are also due to the members of my committee, Dr. T.A. Marsland and Dr. S. Cabay for their suggestions and criticism.

Without the numerous hints obtained from my fellow graduate students and the help provided by Lloyd Benbow this task would have been much harder.

The financial assistance provided by the National Research Council of Canada and St. Francis Xavier University, in the form of scholarships, and the Department of Computing Science in the form of teaching assistantships is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 The Discipline of Information Retrieval

In the history of human progress the following situation occurs quite frequently. At a certain point in time when some technological advance is made, or a new intellectual scheme is developed as an answer to a specific problem, its application is seen only with reference to the task for which it was developed. However, concurrently, or shortly afterwards, another problem from a different and completely unrelated field arises and is found to be unsolvable by use of traditional methods known to workers in that field.

After many unsuccessful attempts to extend the traditional methods it is realized, often by chance or accident, that the advance made within the first discipline may be adapted to solve the new problem. This potential was not realized by the original architects and designers of the original advance, either because of their unawareness of the other field, or because the existence of the other problem was still hidden in the future. Quite frequently this transfer of techniques between fields adds considerably to the body of knowledge and points to new directions which, when followed further, often spawn new disciplines.

The emergence of information retrieval as a separate discipline is an example of the above pattern. The technological advances in the first half of the twentieth century led to the formulation of more complex numerical problems which caused a demand for fast calculators. The computer was developed to answer this demand, and its use was initially restricted to the solution of numerical problems.

In the postwar period it was realized that the rapid growth of the sciences had created a hitherto unknown phenomena which was called the "information explosion". The regular custodians of recorded knowledge, the librarians, were confronted with an unmanageable flood of information and requests for it. Vannevar Bush [10] was the first to draw attention to the fact that the computer had potential in the management and dissemination of information in addition to its already accepted potential for numerical problems.

Attention quickly focused on the new application, and information systems were conceived and developed. Some technical workers familiar with the new machines were already aware of their personal needs for quick and easy access to information, and hence were ready to work on the development of mechanized information retrieval. Since that time the development of information systems has paralleled that of computing systems [4], and has even influenced the design of such systems. The combining of computer science skills with the knowledge available in library science has led to consequent development of new techniques and to the growth of the field of information retrieval.

Information retrieval is defined by Salton [71] as the "field concerned with the structure, analysis, organization, storage, searching and retrieval of information". In this discipline there have developed methods, algorithms, and complete systems to handle the vast amounts of information that is required by today's technological society. The important concept of a database as a collection of related information has emerged, and at present a lot of consideration is being given to database problems. But the movement towards the design of the perfect form of a database is still in its infancy [12] and therefore many open questions remain. Among them are questions concerning database storage and organization, integration of different databases, and development of efficient procedures applicable to searching of different forms of databases.

1.2 Research Objective

The work described in this thesis relates to storage and management considerations that arise in the design of databases for information retrieval. A particular approach to database design is suggested, and is studied with regard to its feasibility for practical application.

The aim of the investigation is to provide sufficient data and experimental evidence for an evaluation of the proposed techniques as well as to indicate the inherent problems that must be considered during the system design. Formulae are developed to form a basis for extrapolation.

Information theoretic techniques, and considerations based on linguistic concepts, are used as a basis for making certain decisions concerning the system design. It is not intended to develop detailed theoretical analyses of all aspects of the proposed approach. It is hoped, however, that the basic theory may be presented in such a form as to provide a firm foundation for the approach adopted and to justify the gathering of the required experimental data.

1.3 Environment of the Study

Certain basic concepts and assumptions form the intellectual requirement for any study, providing the foundations and limiting the scope of the work. They include the set of axioms on which all further deductions are based. It is important to explain the basic concepts and assumptions in some detail before proceeding with the study.

The special area of information retrieval to which the present investigation relates is that which deals with retrospective document retrieval systems that involve very large databases. This implies the use of special methods to handle and retrieve the information. The systems of particular interest are those intended to be operated on-line. This requires special emphasis to be placed on two particular aspects. These are the size of core resident system elements, and the number of disk accesses for the system in operation. The problem of updating the system is not considered in the present study although it may prove to be important in some instances. Also, considerations of database structuring to allow adaptive retrieval are not included in the present study. Such adaptive retrieval might be through user feedback or other means.

The basic elements of most existing retrieval systems are the words as in common usage. The main concern of the present study is to investigate how the use of alternate basic elements called "fragments" may affect system organization and performance. Thus the scope of the present research may be summarized as consideration of: " The effect of fragment use on organization and performance of a static, on-line retrospective retrieval system".

The value of such a study could be questioned with regard to its validity if a certain basic "stability" assumption were omitted, and therefore attention will be first directed to the implications of this assumption. Stability and homogeneity of the database over a period of time is postulated throughout the present study. It is supposed that a sample of the database can be considered to be representative of the vocabulary of the whole. With this assumption the validity of any results derived from the sample can be extended to apply to the whole database provided the sample is sufficiently large. According to

Mandelbrot [48] we must consider:

"... samples of discourse long enough to allow one to speak of probabilities as limits of frequencies and short enough to allow one to assume that these probabilities have not changed between the beginning and end of the sample."

The assumption implies that for different, but sufficiently large, samples of equal size chosen from the database, only small differences are expected in the rank frequency distribution of words and in the distribution of word lengths, regardless of the point in time at which the samples were taken.

The work of Long et al. [41] provides some justification for the assumption made above, even though their investigation deals only with a very special vocabulary. However, strong support for the validity of the assumptions is also provided by Lynch et al. [46]. Their examination of the INSPEC database showed that over a three year period the character and n-gram distribution showed very little variation. They report: "... interfile differences are minimal and no significant trends apparent".

This evidence is considered to be the support for the justification of the assumptions which will be tacitly present throughout the following chapters.

CHAPTER II

BACKGROUND AND SURVEY OF RELATED LITERATURE

The purpose of this chapter is to relate the research of the present study to work already published and documented in the open literature. Only material relevant to the present study will be discussed here since a general introduction to information retrieval is contained in a number of books [71],[51],[3]. The present chapter also provides most of the background material, definitions, and equations on which subsequent chapters depend. Certain aspects are discussed in detail and some of the particularly relevant papers are studied closely. The implications of their content are considered, together with certain concepts chosen from linguistics and information theory.

2.1 Linguistic Elements

The design of any system that involves natural language processing must also take into account certain language characteristics. If the text in a database is considered to form a subset of natural free language then the relevant characteristics may be used to advantage in the design of a retrieval system. The properties of interest in the present study are all of a statistical nature and can be borrowed from studies of statistical linguistics.

Most of the required properties may be derived from a rather simple language model. According to Chomsky [16] an appropriate model is based on the familiar conception of language as a particularly simple type of information source, namely a finite Markov process. This finite state Markov model has been the subject of frequent linguistic studies, one of the most recent being that of Damerau [21]. Despite certain obvious deficiencies of the model, it proves sufficient to introduce those elements of statistical linguistics that are required in the present study.

The oldest, and perhaps most widely known, vocabulary relation was brought to light by Zipf [95] and has been the subject of many discussions. The so called "Zipf's law" may be summarized as follows:

If the D different words (types) in a sample of T words (tokens) of free text are ranked in order of decreasing frequency, it is found that the following empirical relationship is approximately true:

$$f(r) = C/r, \qquad 2.1(1)$$

where f(r) is the frequency of the word of rank r and C is a constant determined by the relation

$$T = C * \sum_{1}^{D} r^{-1} \qquad 2.1(2)$$

A somewhat more complicated equation

$$f(r) = C*(r + V)**-B$$

has been proposed by Mandelbrot [47] and has been explained as the result of a minimization process in which the "cost" of a word is to be minimized. The "cost" is envisaged as encompassing everything which affects the expense of transmitting the word. It may be visualized as the number of digits necessary for transmission.

In Mandelbrot's equation the constants V and B are language dependent and indicate the richness of the vocabulary, while C is determined by an equation similar to 2.1(2). Miller [52] has shown that 2.1(1) may be derived from a simple two state Markov model, subject only to the assumption that the probability of letters is approximately 0.8 and that the probability of a space is approximately .2.

It must be mentioned that the Zipf law has been severely criticized by many writers, particularly Herdan [31]. Despite such criticisms the Zipf law is useful because of the simple form of equation 2.1(1) and because it has been found to give a relatively good fit with measured data. Hence it will be used throughout the present study.

To bring equation 2.1(1) into manageable form a well known approximation for the sum in 2.1(2) can be used to give the formula

$$C = T/\ln(D + g), \qquad 2.1(3)$$

where g is Euler's constant and ln denotes the logarithm to the base e. This yields the following equation for the relative frequencies p(r) of a word of rank r:

$$p(r) = 1/\ln(D + g) * r^{-1} \qquad 2.1(4)$$

Booth [5] has noted that equation 2.1(1) may be used to calculate the number I(n) of words that occur n times in a sample of text. For small values of n the I(n) are given by:

$$I(n) = C/(n(n+1)) , \qquad 2.1(5)$$

which implies that approximately 50% of all words occur only once, 16% twice, and so on. The implications of this equation regarding the storage requirements of an inverted index will be described in Chapter III.

The Zipf law is only one of many distributions which have attracted the attention of linguists. Among these relations is that which describes the distribution of word types and tokens as a function of word length. Such distributions have been thoroughly investigated by linguists [31],[53],[14] and a simple equation has been shown to satisfy most of the data. If the relative frequencies of word types, or tokens, of the same length are plotted against the logarithm of their respective length, a normal distribution is observed. This has been experimentally verified by Carroll [15] for a sample of one million words. The relative frequency of word types or tokens of length L, is found to be given by:

$$p(L) = (1/s\cdot\sqrt{2\pi})*\exp(\ln(L) - u)^2/2s^2) \qquad 2.1(6)$$

where the constants s and u have different values for types and tokens. Since these distributions were observed for samples of free text it is of interest to see if they also hold for the database vocabularies used in the present study.

Studies of statistical linguistics have also provided a relationship between the number of types and tokens. This is the well known type-token relationship which has been explored by various investigators [48],[31] and has been found to be valid for various databases [74], [64]. It states that the number of word types D is related to the number of tokens T by an equation of the form

$$D = K * T ** b \qquad 2.1(7)$$

where K and b are vocabulary dependent constants and $0<b<1$. Since this equation relates the growth of the vocabulary to the length of the database it has far reaching implications for the design of retrospective systems. A discussion of the influence of database characteristics, particularly the Zipf law, on system performance is given by Lowe [44], and in more detail by Siler [83]. This aspect will be discussed again in Chapter III.

In the present study considerable attention is focused on the vocabulary of the database, which may vary considerably among different databases. One database might use a few words very frequently, while another might contain a large vocabulary of seldom used words. For this reason linguists criticize the Zipf law as being unsatisfactory for the characterization of a text sample, since the only parameter available, the constant C of equation 2.1(3), is a function of the sample size. Good [27] has proposed that the repeat rate, defined as

$$re = \sum p(r)^2 ,$$

may be used as a vocabulary characteristic since it is independent of sample size. According to Herdan [31] an unbiased estimate of re is given by:

$$re = (\sum I(r)*r^2 - T)/T(T-1), \qquad 2.1(7)$$

where I(r) is the number of different words that occur r times in the sample. This quantity, together with the average word length, will be used as a characteristic of the vocabulary throughout the present investigation.

2.2 Information Theory Contributions

As important as the study of related linguistic elements is a knowledge of some of the concepts of information theory. These concepts cannot be neglected as they are valuable for the development of some of the theoretical background for the present study. Furthermore, information theory provides theoretical lower limits for some quantities that are of interest to the present study. A knowledge of these limits is valuable since they provide a base for absolute comparisons.

Information theory was developed by Shannon [81] and was soon applied to different fields of science and other areas. Shannon related information theory to linguistics [82] and his results laid the foundations for a new approach to language and its treatment. The definitions and equations that form the basis of information theory are readily available in the literature [65],[61]. Of particular relevance to the present study is the book by Abramson [1].

The information of an event x that occurs with probability p(x) is given by:

$$I(x) = - ld(p(x)) ,$$

where ld denotes the logarithm to the base 2. The average information content of a series of n mutually exclusive events x(i) having a probability of p(x(i)) is defined as

$$H = -\sum^{n} p(x(i))*ld(p(x(i))) . \qquad 2.2(1)$$

This quantity has been called the "entropy" because of its analogy to the definition in physics. The unit of measure for H is the bit. It is well known that H is a symmetrical convex function with its maximum at the point p(x(1))= ... = p(x(n)) = 1/n . The entropy H is an important quantity in any system concerned with information handling and transmission. Since any retrieval system can be considered as a special case of a general communication system as shown in figure 1, the properties and parameters of such a system play a significant role in a retrieval system. A discussion of such a system and its respective entropies is therefore relevant to the present study.

The communication system described in figure 1 consists of five parts which perform various functions. The source S generates n different characters X = { X(1),..., X(n)} with probabilities p(j), j= 1,..,n which are passed to the channel encoder. The channel encoder converts source characters, or blocks of characters, into the symbols or strings of symbols needed for transmission through the channel. Let T = { T(1), ... T(t) } be the set of symbols used for transmission. These symbols form the transmission

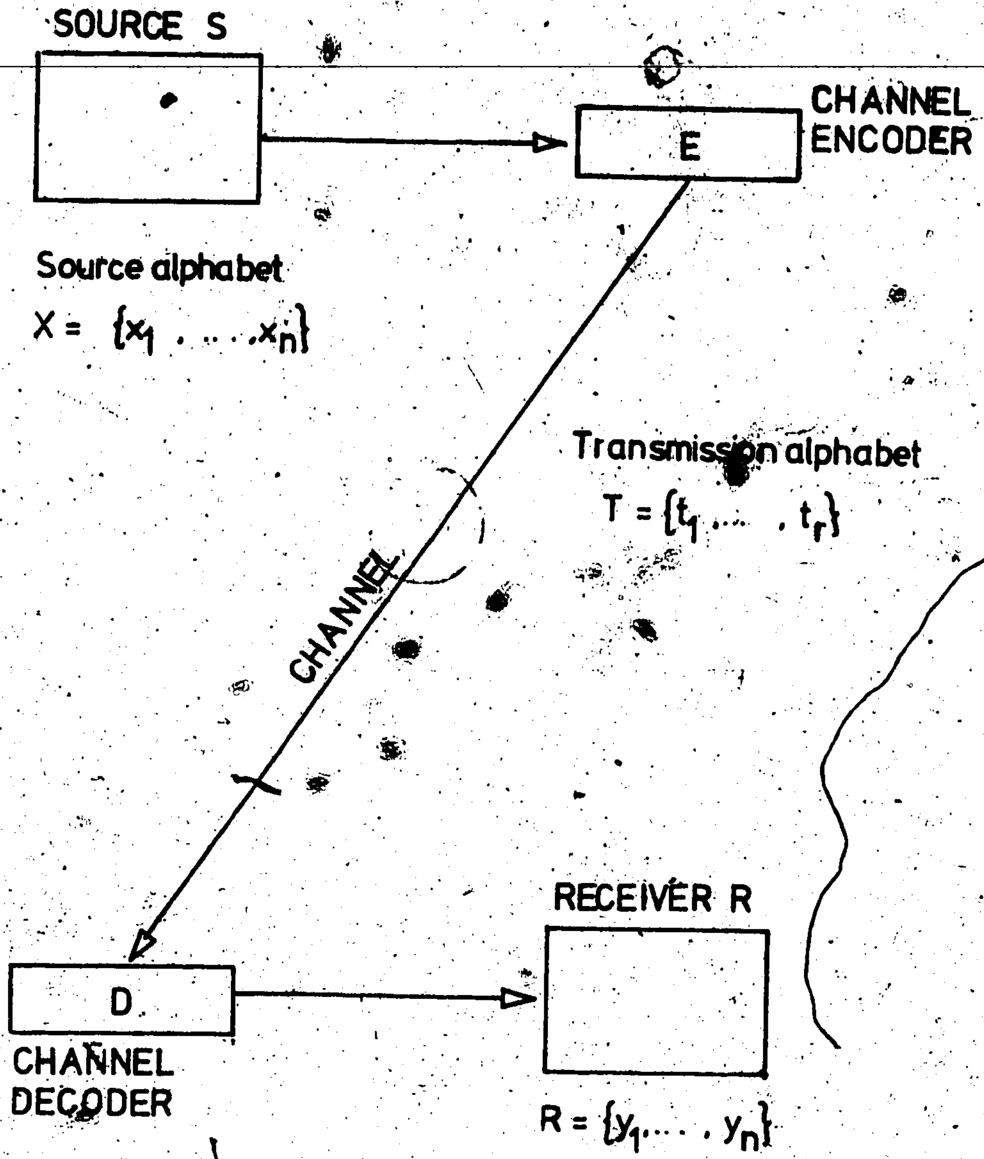


Figure 1
General Communication System

alphabet. The channel encoder E performs a mapping f of a varying number of source symbols into the transmission alphabet T.

$$f: X \longrightarrow T$$

The group of transmission symbols for one block of source symbols is called the corresponding codeword for that block.

The channel C is a medium for information transmission and can have different properties. The channel decoder D performs the inverse mapping f^{-1} from the transmission alphabet into the blocks of source symbols. If the mappings f and f^{-1} are one-to-one then in the terminology of information theory we have a noiseless channel; otherwise the channel is said to be noisy. The noise characteristics of the channel are usually specified by an n x n matrix of conditional probabilities $p(y(j)|x(k))$ which indicate the probability that $y(j)$ was received when it is known that $x(k)$ was sent. Obviously for a noiseless channel $p(y(j)|x(j)) = 1$ for $j = 1,n$ and $p(y(i)|x(j)) = 0$ for all i and j except when $i = j$. The following three quantities, all different entropies, are sufficient to characterize the transmission between source and receiver.

H(X)	Average information at the source defined by equation 2.2(1)
H(Y)	Average information at destination defined by equation 2.2(1) with y(i) replacing x(i)
H(X|Y)	Measure of information about the source when it is known that Y is received. This is sometimes

called equivocation, and indicates how well the input can be recovered from the output. It is defined by:

$$H(X|Y) = -\sum \sum p(x(i),y(k))\, ld(p(x(i)|y(k)) \qquad 2.2(2)$$

It may be seen that a knowledge of the above three entropies implies the availability of the probability distribution at the source and receiver, the joint probability matrix p(x(i),y(j)) and the noise matrix which specifies the conditional probabilities between source and receiver. The average mutual information transmitted by the system, sometimes called the "transinformation", is given by:

$$I(X,Y) = \sum \sum p(x(i),y(j)) * ld(p(x(i)|y(j))/p(x(i)) \qquad 2.2(3)$$

By virtue of equation 2.2(2) the equation for I(X,Y) may be rewritten as

$$I(X,Y) = H(X) - H(X|Y). \qquad 2.2(4)$$

Assuming that the transmission time for all transmission symbols is the same then the capacity C of a channel may be defined as the maximum of the transinformation I(X,Y) for a given noise matrix. The maximization is with respect to all possible sets of probabilities that could be assigned to the source. Thus channel capacity is linked to the transinformation I(X,Y) through the relation

$$C = MAX\ (\ I(X,Y)\ ). \qquad 2.2(5)$$

The difference between the actual rate of information transmission and its maximum possible value C is defined as the absolute redundancy, while the ratio of absolute redundancy to channel capacity C is called the relative redundancy. An important parameter that occurs in the subsequent sections is the efficiency of transmission Te defined as

$$Te = I(X,Y)/C. \qquad 2.2(6)$$

The above few definitions constitute all the elements of information theory needed in the present study. In the special case of a retrieval system an assumption may be made to allow the general equations to be simplified and brought into a more manageable form.

The assumption which allows the reduction in complexity is that the channel is considered noiseless. This implies a noise matrix in diagonal form so that $p(x(i)|y(i)) = 1$ and $p(x(i)|y(j)) = 0$ for all $i,j = 1, \ldots, n$ $(i \neq j)$. It may be seen from equation 2.2(3) that this reduces 2.2(4) to

$$I(X,Y) = H(X). \qquad 2.2(6)$$

Using the fact that H(X) has a maximum if all source symbols are equally probable the channel capacity C is given as

$$C = ld(n). \qquad 2.2(7)$$

and 2.2(6) can be reduced to

$$Te = H(X)/ ld(n). \qquad 2.2(8)$$

With these basic definitions the operation of the encoder can be studied in more detail and concepts from coding theory can be applied.

The encoder associates with a letter, or a sequence of letters, from X an element from the transmission alphabet; the element is called a codeword. This codeword is used for the transmission through the channel, and commonly consists of a sequence of binary digits. Since a certain cost is generally assumed for all the symbols going through the channel, the encoder is expected either to reduce overall transmission cost or to achieve more reliable error free transmission. Shannon has proved that H(X) is the minimum average code length possible for a transmission of symbols with a distribution given by p(x(i)), i=1,n. The average code length H(X) results only if the codewords are expressed in the most compact code. Let b(i) be the length in bits of the codeword for the element x(i) whose probability of occurrence is p(i). The average code length L is given by

$$L = \sum p(i)*b(i) \quad , \qquad 2.2(9)$$

and using Shannon's result that H(X) is the minimum value obtainable for L, the efficiency Ce of a code is defined by

$$Ce = H(X)/L. \qquad 2.2(10)$$

The redundancy of the coding is given by

$$RD = 1 - Ce$$

and a code which minimizes RD is called a minimum redundancy code. A redundancy of zero is obtained if b(i) = -ld(p(i)). However, in practice there is the limitation that the codeword must consist of an integral number of bits, and hence the minimum practical value of b(i) is

$$b(i) = ceil(-ld(p(i))) \quad , \qquad 2.2(11)$$

where ceil(x) denotes the next integer greater or equal to x. The resulting minimum value of Rd reduces to a value of zero only if the p(i)'s are inverse powers of two.

The problem of finding a minimum redundancy code for a given probability distribution has been solved by Huffman [33] by a well known combinatorial type algorithm. The more complicated problem of determination of a minimum redundancy code, which preserves a given order of input has been solved by Gilbert and Mooore [26]. The codewords generated by either algorithm are of variable length and are uniquely decipherable or separable since they possess the prefix property for codes.

The literature concerned with coding is very extensive since it is not only important in the study of communication systems, but also for the construction of logical and arithmetic circuits.

Coding and information theory can now be applied to linguistics since language can be considered to originate from a simple Markov type information source. The linguistic interpretation of the two basic definitions of entropy and redundancy was given by Shannon [82]. He indicated that:

" The entropy is a statistical parameter which measures in a certain sense, how much information is produced on the average for each letter of text in the language. If the language is translated into binary digits in the most efficient way, the entropy H is the average number of binary digits required per letter of the original language. The redundancy on the other hand measures the amount of constraint imposed on a text in the language due to its statistical structure."

Shannon also has estimated[7] that, when statistical effects extending over not more than eight letters are considered, then the entropy is approximately equal to 2.3

bits per letter.

This reduction in entropy per letter can be achieved by coding blocks of letters for transmission through the channel instead of coding only the single letters. The choice of language elements for coding is made by the system designer. He can choose either fixed or variable length blocks of letters depending upon his specific requirements. A natural choice of language element to be coded is the word defined as a string of characters delimited by blank spaces. If a codeword is transmitted, and words are the language elements used for coding, a blank is implied to follow each word in the decoded output.

For transmission based on words as language elements a relation between codes and words is required. Such relations can be provided through use of a code book, or code dictionary, that lists words together with their respective codes. If sender and receiver have the code books, and the channel is noiseless, then error free transmission is guaranteed provided the channel capacity is not exceeded. The entropy of a text with T words can readily be calculated by combining the Zipf law of equation 2.1(4) with the definition of H in 2.2(1). It yields:

$$H(T) = ld(\ln(D + g)) + ld(D)^2/(2*\ln(D + g)). \qquad 2.2(12)$$

The corresponding transmission efficiency is given by

$$Te = H(T)/ld(D), \qquad 2.2(13)$$

since there are D different words in the code[illegible]onary. A minimum redundancy code can be constructed f[illegible] words by applying Huffman's algorithm to the relative frequency distribution of the words.

The above considerations have far reaching implications for a retrieval system, since the problem of document storage must be solved before successful operation of the system.

Elimination of coding redundancy is clearly desirable. The redundancy inherent in the machine oriented character code representation can be reduced by use of Huffman codes for the letters, but can also be improved by other schemes, among them word coding.

The reduction of storage requirements by elimination of redundancy is called compression. A measure of compression is the compression ratio Cr as defined by Davidson [22]:

$$CR = \frac{\text{Length of original string}}{\text{Length of compressed string}} \qquad 2.2(14)$$

The inverse of CR is used more commonly in information retrieval since it expresses the fraction of the original string needed to store the compressed string. It will be called compression and will be denoted by ICR. Because of the great importance of compression in the operation of large retrieval systems a closer look at various compression methods seems desirable. The popularity of text compression can be seen from the large amount of literature available. One of the first surveys were those of Nugent [56] and Ramamoorthy [63]. More recent ones are those of Mulford et al. [54] and Byrne and Mullarney [11]. A bibliography of papers related to compression is provided by Ruth and Villers [68].

2.2.1 Nonsingular Compression

Compression methods can be divided into two groups. The first group to be discussed includes all ~~schemes~~ of the type which Nugent [56] has called nonsingular. These are the methods that can map more than one input word on to the same codeword, so that when given a codeword there is ambiguity in recovery of the input. This definition of nonsingularity is somewhat at variance with the mathematical use of the term to describe singular matrices and transformations.

In many such schemes the redundancy of information is reduced by discarding certain characters of the word to be compressed. Clearly this results in a loss of information. However, nonsingular methods have many applications and include many well known algorithms that reduce the input words to a fixed length key for use in a particular situation. Surveys of the different schemes can be found in publications by Bourne et al. [7] and Nugent [56]. The method used to eliminate selected letters depends on the applications, and these can be divided into four areas as described below.

For example, a common problem in the publishing and banking industry is the generation of a set of keys to be used for subsequent searching of the database, with the order of the input words preserved in the keys. A solution is to truncate the words to fixed length, usually by dropping the rightmost lettters. The uniqueness of the resulting keys as a function of the key length has been studied by Lowe [45] and the effectiveness for retrieval by Siler [83] and Lowe [43].

A second application concerns the use of a set of retrieval keys whose derivations from the natural words remain invariant under a certain amount of incorrect spelling of the input. Such keys have application in airline reservation systems. The solution is normally by use of Remington Rand's SOUNDEX code or vowel elimination as described by Bourne [7].

A similar problem, but with a different hypothesis about the input arises in library automation. A set of keys from accurate bibliographic records must be created with the objective of maximum discrimination between similar entries. Algorithms to achieve this objective have been proposed by Nugent [57] and Ruecking [76] and their retrieval performance was tested by Libetz et al. [40].

A further group of nonsingular compression methods has been designed with the objective that the resulting set of file keys shall exhibit high human readability and high discrimination between similar words. Dolby [23] proposed an algorithm to solve this problem, but more common are Alphacheck methods, which consist of truncation plus an additional check character determined from the full input word as described by Bourne [7].

The main application for nonsingular compression methods is in the generation of keys for retrieval. The methods are not suited to store document records in compressed form when it is required to have an exact representation without information loss. Record and file compression is usually achieved by singular compression in which there is a one to one relation between input and codeword, so that the relation may be inverted uniquely.

2.2.2 Singular Compression

For a retrieval system that involves a large database it is highly desirable to compress the records, the larger the compression ratio the better. This is the main application for singular methods. The Huffman code is the most advantageous to use for compression. It was shown by Gilbert and Moore [26] that the Huffman encoding has a code length which is less than or equal to that of any other decipherable code. The first proposal to use Huffman codes for document compression was made by Schwartz [76], and a program to generate the codes for a given distribution of words is described in [77]. The effect of imposing additional constraints on the codes was investigated by Schwartz [78]. Practical results regarding compression of business files are given by Ruth [69] and for system files by Wells [92].

The one to one relation between words and codes may be specified through a stored dictionary of all different words together with their associated codes. These codes are then used for storage of the database in compressed form. The coding process requires the location of the code for the input word and the insertion of this code in the compressed database. Decoding is more tedious, since the variable length codewords must be decoded by a tree search. This makes the use of Huffman codes awkward, and the task of storing and updating the dictionary is not to be underestimated.

Since the handling of variable length codes in computers with a fixed word size is clumsy, it was proposed by Heaps and Reid [64] to use restricted variable length codes. A bit string of unit length suitable for convenient machine processing is chosen. An obvious choice of unit length is six or eight bits. Codes are constructed by concatenation of the units, so that the code length is always a multiple of the basic unit length. A detailed description of the use of these codes, together with a thorough storage analysis is given by Heaps [29], while particulars about the program design for coding and decoding are reported by Thiel and Heaps [87].

The approach to compression taken in the National Bureau of Standards SOLID system is quite different as evident from the descriptions [49],[24]. The SOLID system was designed to be independent of any linguistic influence, and hence words of natural language were not chosen as the language elements for compression. Variable length bit strings of high frequency are replaced by fixed length codes. The compressed database consists of a mixture of both codes and uncompressed bit strings. The latter are the bit strings whose low frequency does not warrant the introduction of a codeword. A dictionary is also used in the SOLID system. It relates bit strings to corresponding codes. The dictionary can be limited in size, because the minimum frequency of occurrence for a bitstring can be varied as needed before a code is introduced for it.

Yet another approach has been taken by Snyderman and Hunt [84] and by Schieber et al. [73]. They propose to assign the unused EBCDIC combinations to the most frequent digrams and to store the database by replacing these digrams with the code. Schieber rejects the use of trigrams or other n-grams because he feels that the increase in processing time is not justified by the improvement in compression. This method preserves the advantage of fixed length codes. It allows simple decoding by indexing into a table, and still achieves good compression. Schieber claims a saving of about 40% in the storage requirement, which is close to the theoretical limit of 50%.

When picking a singular compression method the choice must not only be made among the three schemes described, but also among the possible language elements. It is possible to choose a linguistic unit, such as the word, to use fixed length character strings, or to emphasize the bit-string as in the SOLID system. Schwartz [77] proposed to use word pairs of high frequency as language elements and he reports that considerable compression was achieved for a highly specialized vocabulary. The investigation of variable length strings as language elements will be pursued in the present study and the literature relevant to this approach is referred to in the next section.

2.3 Work related to Fragments

With the arrival of a large number of computerized search services several investigators have given consideration to the use of variable length character strings and its effect on system operation. Such efforts have followed two different directions. One was to facilitate the use of truncated search terms, while the other tried to study the suitability of these strings for compression.

The first reference to the term "fragment" apppeared in a paper by Colombo and Rush [19] in connection with current awareness searches of Chemical Title tapes. Their retrieval system is based on a sequential search and permits truncated words to be used as search terms. In a sequential search the search time is directly proportional to the number of characters in the search term. To reduce this time they tried to isolate in each word the shortest possible character string which uniquely identifies the word. These variable length character strings were named "word fragments". To assist the system user in his use of the fragments they generated a fragment-word list in which each fragment of length greater than three is listed together with all words that contain the fragment. A similar list called a KLIC (KeyLetter In Context) was published by Kent [37] with the intention of showing the effect of the use of left truncated search terms.

Variable length character strings as the basis for transmission of data were first investigated by White [93] with special emphasis on the elimination of redundancy. His method includes the construction of a dictionary to contain the most frequent words, bigrams, and longer n-grams. Each dictionary entry was to be coded for transmission purposes. The main feature of the paper is the suggestion that the encoder collect statistics on the coding performance and that he use the results to modify the dictionary and make it closer to optimal. The feasibility of this suggestion was tested by a simulation through which a compression of the original text to about [illegible] or 70 percent of its original size was predicted. The idea of an adaptive dictionary is feasible for transmission where the codewords are decoded immediately after transmission. It is unsuitable for compression in a retrieval system since there may be a considerable time lag between the input of the coded string and the decoding. A change in language elements for the codewords between coding and decoding would have a disastrous effect.

Word fragments and data compression were first associated by Walker [89]. He introduced the idea of using variable length character strings, which he calls X-grams, to store names in compact form by concatenation of codes for the X-grams. The set of X-grams form a dictionary in which each entry is associated with a fixed length code. The X-grams in the dictionary are selected by an iterative algorithm which takes into account both frequency and length.

A recent suggestion by Clare et al. [17] also proposes the use of variable length character strings. However, these strings are not limited to be elements of words, provided they are part of the record. For this reason they can be called "text fragments". The term fragment includes word and text fragments throughout the remainder of this study. Text fragments may contain embedded blanks, and obviously the set of word fragments is a subset of the set of text fragments.

Clare et al. place an interesting additional restriction on the text fragments to be included in a dictionary, namely that the dictionary fragments should occur in the database with approximately equal frequencies. These dictionary fragments are intended to be used as the indexing, compression and retrieval elements. An algorithm is described which reduces the frequency of occurrence of the fragments to lie below a certain threshold. Only text fragments with a frequency below that threshold are included in the dictionary. The effectiveness of these elements as text units has been investigated by Barton et al. [2]. The proposal by Clare et al. has sufficient merit to warrant a closer look, especially at the implications of the equifrequency property.

If the text fragments are to be used for indexing purposes, the equifrequency property of terms used for indexing is very desirable. The investigation of Svenonius [86] and Salton [70] prove this. Salton makes the following observation in his statement of the principles of thesaurus construction:

" Each group of related terms should account for approximately the same total frequency of occurrence of the corresponding words in the document collection. "

Since text fragments would satisfy this requirement, they would indeed be the ideal indexing elements. A somewhat different result was obtained by Zunde and Slamecka [96] using an information theory model, but as Svenonius notes it is questionable whether "information in the sense of information theory can provide an adequate measure for evaluating the distribution of index terms".

The characteristics of good (discriminating) and bad (non-discriminating) index terms have been studied recently by Yu [94] and Salton [72]. Good and bad are defined with the help of the frequency distribution of index terms in the database.

It is worthwhile to examine Clare's proposal from an information theoretic approach, despite the criticism of information theory in connection with the evaluation of indexing systems. One of the proposed uses for fragments is data compression and coding, and for this information and coding theory provide the necessary measures.

Assuming that the dictionary consists of ND approximately equifrequent fragments, it can be seen from equation 2.2(11) that minimum redundancy codes would all have a common code length of C = ceil(ld(ND)). Since the dictionary size can be chosen by adjusting the threshold it is possible to let ND be a power of two. Let the entropy for the ND dictionary fragments be H, then the measures Ce and Te are given by:

$$Te = H/ld(ND) \qquad 2.3(1)$$

$$Ce = H/\,ceil(ld(ND)). \qquad 2.3(2)$$

If ND is a power of two then Ce and Te can be combined into a ratio, called the efficiency E, defined by

$$E = H/ld(ND). \qquad 2.3(3)$$

This quantity E as defined above will be used as a measure of equifrequency. It has the value of one for a uniform distribution and is less for all other distributions. It must be remarked that because H involves logarithms, E is a ratio of logarithmic measures and hence a small increase in E indicates a considerable improvement toward a more uniform distribution.

The critical point in the use of equifrequent fragments for compression (as proposed by Clare et al.) can easily be overlooked. Their dictionary consists of all fragments that occur with a frequency less than a certain threshold value, and such fragments have considerable mutual overlap. The compressed database consists of the codewords for the fragments and a record is represented by the concatenation of these codewords. Overlap between the fragments of two successive codewords cannot be tolerated for two reasons. First it would be necessary to specify the amount of overlap between the fragments of successive codewords, and secondly there would be a substantial increase in the storage requirements for the compressed database since certain characters within the record would be stored more than once. If, however, the database is coded by non-overlapping fragments the resulting distribution of fragments used for coding will be completely different from the distribution of overlapping fragments in the dictionary. For these two reasons it is necessary to code the database by use of non-overlapping fragments and take this into account when selecting dictionary fragments. The reduction of overlap among dictionary fragments will decrease the size of the dictionary.

Choosing fragments with a frequency less than a threshold (as done by Clare et al.) must also be questioned with regard to efficiency of compression. It is not advantageous to replace a fragment of low frequency by a code because of the fairly high cost of including it in a dictionary which must be kept in an easily accessible, and therefore more expensive, storage medium. For compression purposes it is more desirable to code high frequency fragments and to exclude the low frequency ones. This approach is taken in the present study by considering only high frequency fragments. Furthermore the reduction of mutual overlap between fragments is considered to be a major factor during construction of the fragment dictionary. The actual compression of the database using a given dictionary is a significant problem, since the equifrequency property should be preserved.

Wagner [90] has treated the problem of minimum text storage in connection with work related to compiler construction. He was concerned with the storage of messages formed from a specified phrase dictionary. He considered representation of text by the use of phrases from a given dictionary, the particular set of phrases for a message chosen to require minimum storage space. He provided a dynamic programming algorithm to generate the space optimal representation [91]. It is interesting to note that he did not investigate the problem of determining the set of phrases which would yield maximum compression.

The study of character strings and fragments has recently received the attention of several investigators. Onderisin [58] proposed to perform a dictionary lookup on the least common bigram of the term to be located. In comparison to the dictionary lookup for the initial bigram the method of Onderisin requires a reduced number of lookups in order to find the term in a suitable arranged list. Schipma [74] analyzes fragments with regard to the inversion of files in order to study the problem of truncated search terms in retrospective systems. Rickman and Gardner [66] showed that bigrams could be used for automatic indexing.

From this short overview it can be seen that units smaller than words have been successfully investigated from different points of view in several areas of information retrieval. It apppears that many more applications can be expected in the near future.

A similar approach based on the use of units smaller than words has been tried in the area of computer speech processing. This is of particular importance with regard to speech synthesis for telephone answering systems [25].

Small phonetic units, in this case syllables, are recorded and stored on a special disk suited for audio recordings. Words and messages are produced by concatenating the prerecorded units with some additional smoothing between neighboring syllables. Such a scheme has produced satisfactory results [36].

The related development referred to above indicates that information retrieval is not the only field in which there is interest in use of fragments smaller than words as the "atoms" of natural language. However, the "atom" may be defined differently for different areas.

CHAPTER III

DESCRIPTION AND PARAMETERS OF RETRIEVAL SYSTEMS

3.1 Current Awareness Systems

Commercial search installations offer a wide spectrum of services to the user. They differ considerably with respect to the number of databases searched, methods used and convenience afforded to the user. Most of the services provide current awareness searches in which questions supplied by the users are compared with the current content of a database whose content is replaced at regular intervals. Such services keep the user up-to-date in his knowledge of new publications and may be made available at small cost. A common procedure used by many providers of current awareness services is to batch all the questions for a one-pass comparison against the database, usually by a sequential search. Only documents that satisfy the question logic are output to the user.

Since search time is proportional to the length of question terms a variety of schemes have been tried to save CPU time. Colombo and Rush [19] have suggested reduction of this time by using fragments as search terms. Onderisin [58] reduced the number of comparisons through the use of a lookup procedure based on the least common bigram.

Question formulation is one of the main differences between services, and for proper assessment of a retrieval system a study of permissible query forms is necessary. The question form described here is a simplified version of the one proposed by Thiel and Heaps [87]. A very detailed explanation with examples can be found in the lecture notes of Heaps [30]. The introductory definition follows Hsiao and Harary [32].

Consider a set of attributes A and a set of "values" V. A record R is a subset of the cartesian product A x V in which an attribute has one and only one value. Thus R is a set of ordered pairs of the form (Attribute,Value). The set of attributes for document retrieval includes author, title, subject heading, publication date, publisher names and so forth. A search query is normally a set of attribute value pairs (AVP) connected by Boolean operators. Only records whose AVP's satisfy the question logic are desired to be output to the user. The basic query structure, without weights for the AVP's is sufficient for the present study. It can be defined by the well known Backus-Naur form:

```
<Attribute>          ::= Element of A, usually specified by a
                         unique abbreviation (AUT,TIT,PUB,...)
<Character>          ::= A|B|C| . . . |X|Y|Z
<Digit>              ::= 0|1|2| ... |8|9
<Alphameric>         ::= <Character>|<Digit>
<String>             ::= <Alphameric>|<String><Alphameric>
<Truncation op.>     ::= $|*
<Rstring>            ::= <String>|<Rstring>$|<String>*
<Lstring>            ::= <String>|$<Lstring>|*<String>
```

```
<Term>              ::= <String>|<Lstring>|
                        |<Rstring>|<Lstring>*|<*Rstring>
<Value>             ::= <Term>|NOT<Term>
<AVP>               ::= [<Attribute>,<Value>]
<Span>              ::= <Digit>|<Span><Digit>
<Logic op.>         ::= AND|OR|ADJ<Span>|PRE<Span>|WITH
<Concept comp.>     ::= <AVP>|<AVP><Concept comp.>
<Concept seq.>      ::= <Concept comp.>|
                        |<Logic op.>(<Concept comp.>)
<Concept>           ::= <Logic op.><Concept seq.>
<Query>             ::= QUE <Concept> END
```

Some explanation about the semantics of these definitions is in order, especially with regard to truncation and logical operators. An AVP represents a Boolean variable. The AVP is considered true if the character string specified by the value can be found in the proper attribute field of the record. If however, the string is negated by the NOT operator, then the AVP is considered true if the string is absent from the attribute field. If the search term is not limited by a truncation operator on one side it is implied that the string must be found in the record and must be delimited by a blank on this side. The truncation operator * indicates that an unlimited number of characters can precede, or follow, the string. The operator $ specifies single character truncation.

The logic operators can be divided into two groups, the regular Boolean operators AND and OR, and the somewhat stronger group of ADJ (Adjacency), PRE (Precedence) and WITH. Operators that are associated with a number that represents a span have the same function as the AND, but with the additional requirement that the terms connected by the operators must be found within a specified number of words as given by the span. The operator WITH restricts the terms to occur in the same sentence. A sample question and its verbal equivalent is shown in Table 1. From this example it can be seen that truncated terms, especially terms with *, can lead to the retrieval of records which are not desired. A record retrieved erroneously because of a truncated search term is called a false hit. The number of false hits gives a good indication of the system's performance.

Standard measures for retrieval systems are precision and recall. Precision is the ratio of retrieved relevant documents to all documents retrieved, while recall is the ratio of retrieved relevant documents to all relevant documents in the database. The problems encountered with precision and recall are discussed by Salton [71]. Various other measures have been proposed, one of the more interesting ones is the expected search length suggested by Cooper [23], which yields a single number as a performance measure.

```
QUE
    AND(
        OR([TIT,TREE$] [TIT,FILE$] )
        WITH([TIT,INFORMATION]
            OR([TIT,RETRIEVAL] [TIT,PROCESS*] ))
END
```

NOTE:

TREE$	includes	TREE,TREES,TREED,TREEN
FILE$	includes	FILES,FILED,FILET
PROCESS*	includes	PROCESS,PROCESSES,PROCESSING, PROCESSIVE,PROCESSION, PROCESSIONAL, PROCESSED

VERBAL:

Find all records in the database which have either TREE$ or FILE$, or both, together with INFORMATION RETRIEVAL or INFORMATION PROCESS* in the title. Furthermore, INFORMATION RETRIEVAL or INFORMATION PROCESS* must occur in the same sentence.

TABLE 1

SAMPLE QUERY AND ITS VERBAL EQUIVALENT

3.2 Retrospective Systems

In contrast to the common availability of current awareness systems is the rarity of search services which use a database that covers a significantly longer period of time. These retrospective systems are of invaluable assistance to any researcher who wishes to investigate a new topic and is interested in all relevant background material. This represents a marked difference to the current awareness systems where a fixed set of questions, the user profiles, are compared to the changing content of a database. Retrospective systems are apt to consist of a large static database with a set of rapidly changing one-shot questions. For these systems a sequential search is not feasible because of the high cost of such a search on a large database. Hence a different search procedure is required.

The size of a large database is one of the major problems. If stored in character form the storage requirements are prohibitive, even for such a cheap and convenient storage medium as magnetic tape. Compression of the database is the normal solution and one of the methods of section 2.2.2 is usually selected for this purpose. Assuming that words are regarded as the natural language elements for coding then a dictionary on random access storage is required for purposes of coding and decoding. The price paid for the compression of the database is hidden in the coding and decoding time, the increase in expensive disk storage and the increase in the number of disk accesses. A detailed description of coding and decoding for restricted variable length codes is given by Thiel and Heaps [87].

In order to formulate the trade-off that results between the decrease of database storage and the increase in dictionary storage, it proves convenient to introduce some basic database parameters. The following discussion is summarized in a paper by Schuegraf and Heaps [75].

Let the database consist of NR records with a total of T words of which D are different. With the assumption that the database consists of TC non-blank characters the average word length AW may be calculated as

$$AW = TC/T . \qquad 3.2(1)$$

The length of the uncompressed database, including the approximately T blanks, in a character representation with CR bits per character is

$$L = (TC+T)*CR . \qquad 3.2(2)$$

If Huffman codes are used for compression, with words as language elements, the entropy HW of the database specifies the minimum number of bits needed to code the language element. The length of the database compressed by Huffman codes is given with HW specified by equation 2.2(12) as

$$LC = (TC/AW)*HW . \qquad 3.2(3)$$

Word coding can neglect the trailing blanks of words in the database since each codeword implies this blank. The inverse compression ratio , indicating the fraction of the original database taken up by the compressed version, is given by

$$ICR = TC*HW/(AW*CR*(TC+T)). \qquad 3.2(4)$$

This value represents a lower bound for the value of ICR and is the value which can be achieved with a Huffman code. Any other code will result in a larger ICR and less compression.

The space on random access storage taken up by the dictionary is given by

$$SD = D * CR * AD \qquad 3.2(5)$$

with AD denoting the average word length in the dictionary. The value of AD cannot be derived from known quantities, but in practice it is always larger than AW, normally by a factor between 30 to 40 percent [74],[88].

It must be mentioned that the actual coding of the database is a one-shot affair, so that the efficiency of the coding procedure is not of prime importance. In contrast the decoding of individual documents is a very frequent operation, and a speedy and efficient decoding process is necessary for good system performance.

The problem of rapid response to a variety of questions can be solved by the construction of an auxiliary file called an index, which normally resides on random access storage. Methods for construction and handling of these files have been discussed by Lefkovitz [39] and in a formal manner by Hsiao and Harary [32]. The best organization for the index depends on the database characteristics and the complexity of the query stream. This has been investigated by Siler [83], Martin [50], Collmeyer [18], and Cardenas [13] who provides comparative data. In general the results show that an auxiliary file organization called an inverted index provides good results for most databases and performs especially well with complex queries. It will be assumed that the auxiliary file in the system to be considered is an inverted index. Its form will be described in more detail in the following part.

Assume the dictionary words to be ordered, say by rank according to frequency, and that each record has been given a unique record identification number (RIN). An inverted index can be defined as a Boolean matrix I of size D x NR with elements I(k,j) = 'True' if word k occurs in document j and 'False' otherwise. For a large database the use of disk for the storage of such a matrix is too expensive, even if the matrix is stored in bit string form.

A different approach to the inverted index is to consider it as a set of lists, in which a list L(k) contains all RIN's of the documents that contain the k-th word. For words that are not repeated within the document, the length of the list is given by f(k) as defined in equation 2.1(1).

The documents that satisfy the query are found by performing the required logic operations on the rows of the inverted index and determining the RIN's of the resulting documents. The documents are then retrieved from the compressed database and decoded. The structure of such a retrospective system is outlined in figure 2.

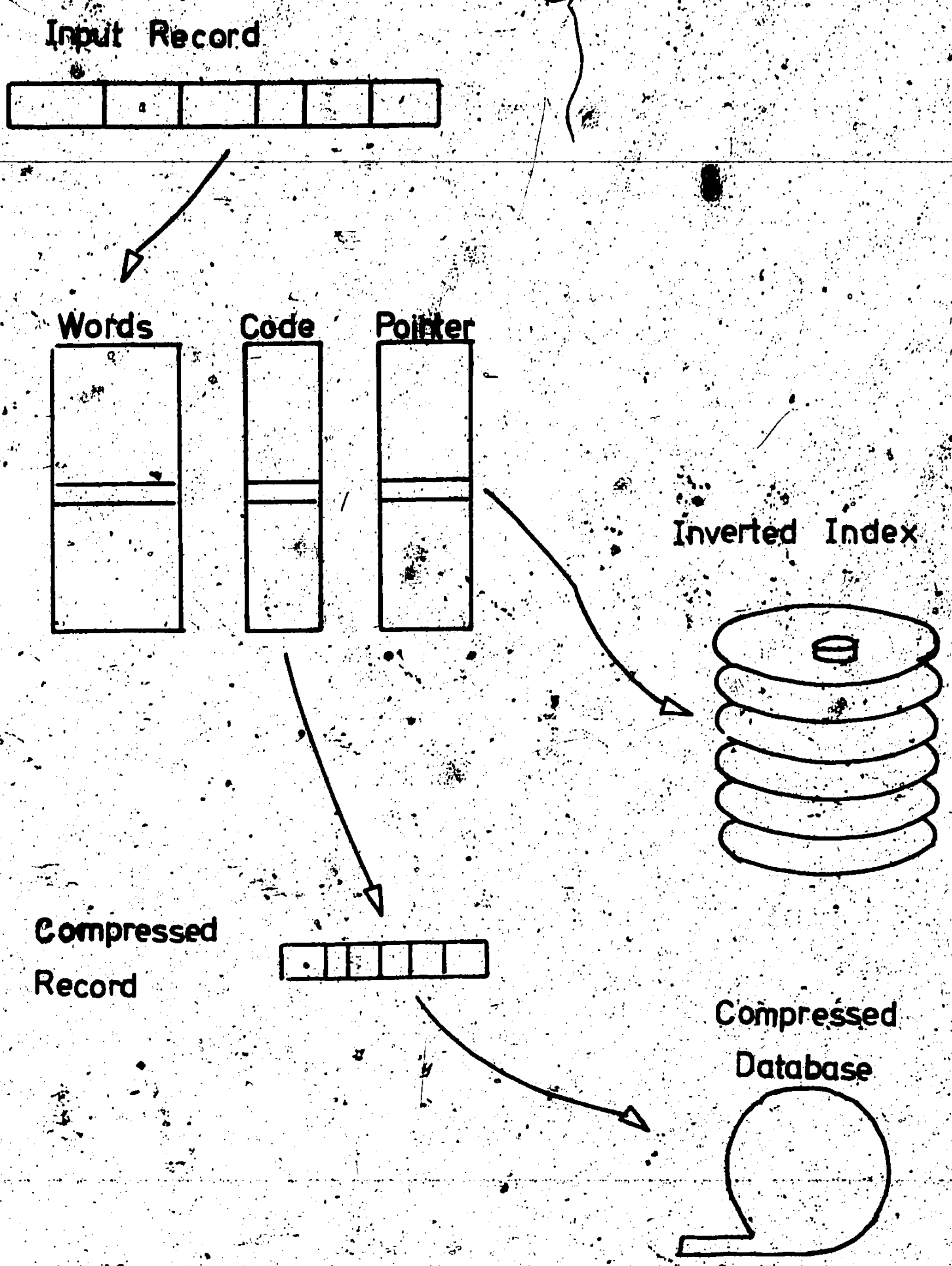


Figure 2

Organization of Retrospective Retrieval Systems

At this point it seems appropriate to remark (that during the search phase of the program that refers to the inverted index) the operators ADJ,PRE and WITH can be treated like the AND, but an additional check of each document is required before the output is given to the user. This is necessary since the index does not contain any information about the order of words in the documents. A sequential search of the documents is used to test if the terms occur within the specified span. In this case the inverted index provides the hooks to retrieve possible candidates which could satisfy the query. When the search logic is restricted to AND and OR the sequential search can be omitted since all retrieved documents satify the query. Obviously the powerful search logic ADJ, PRE and WITH complicates the search procedure by introducing the requirement for a sequential search.

A more serious drawback than the need for a sequential search is the need to store the index on disk in order to guarantee fast response to the queries, and the resulting increase in disk storage cannot be neglected. The information in the database is effectively duplicated in the index and therefore the storage requirements are of approximately the same magnitude. These requirements can easily be calculated if the index is stored in list form. The number of bits required to store a record identification number is

$$B = \mathrm{ceil}(\mathrm{ld}(NR)) , \qquad 3.2(6)$$

and because there are TC/AW entries the required storage for the index is

$$SI = (TC/AW) * R \; . \qquad 3.2(7)$$

This expression may be regarded as a theoretical upper limit for SI since the space occupied by the lists L can be reduced by a combination of bit map representation and RIN's as described by Thiel and Heaps [87].

An obvious reduction in SI may be achieved by eliminating the most frequent words, or words that are not likely to be used for queries. The list of words for which no row of the inverted index is generated is called a "stoplist" and includes words like "the,for,and,of" and many more. The index itself is generated during the compression process and only an additional sort is necessary.

By use of 3.2(5) and 3.2(7) the total disk storage can now be calculated as

$$ST = (TC/AW)*R + D*CR*AD \; . \qquad 3.2(8)$$

The organization of the inverted index is a critical factor in the system and will therefore be given a closer look.

The disk is divided into small sections usually called sectors whose size is often limited by hardware considerations. Each sector, or bucket, consists of a fixed number of bits which can be transferred in one read and hence with a single disk access. Each of these buckets is intended to hold the RIN's of one row of the inverted index. However, the disparate length of these rows as indicated by 2.1(1) seriously affects this intention. Some lists will spill over into more than one bucket while others will not fill their bucket. This makes impossible the calculation of a bucket address for a specific row of the inverted index, so that with each term in the dictionary there must be stored a pointer to the proper bucket. The storage space wasted by unfilled buckets is a more serious matter. It can be kept small by selecting a small bucket capacity; but this choice is limited unless small buckets are processed by software routines which make for cumbersome processing. A good measure for the wasted disk space is proposed by Lowe [44] and is called the memory utilization factor P. It is defined as

$$P = (V - T * R)/T * R \qquad 3.2(9)$$

where V is the actual storage occupied, while T*R is the total minimum amount of storage needed for the index. Assuming a bucket capacity of C*R bits, use of 2.1(11) allows V to be calculated as

$$V = C*R \sum_{1}^{D} \text{ceil}(T/\ln(D+g)*1/(r*C)) . \qquad 3.2(10)$$

It may be noted that P has the value of zero only when no waste occurs, unity when twice as much memory as needed is dedicated, and so forth. Limits for P are given by Lowe [44] and it can be seen that the Zipf distribution is responsible for the large amounts of wasted disk space. If the elements used for indexing are varied so that a more uniform distribution occurs a suitable choice of the bucket capacity C can reduce P considerably.

The use of an inverted index stored on disk results in the requirement for a large number of accesses to the buckets during the retrieval phase, and this is very critical for an on-line system. The different length of the rows of the index is a major factor since for the most frequent terms more than one bucket must be accessed before all document numbers are retrieved. The average number of disk accesses is defined as

$$AC = \sum_{1}^{D} p(r) * \mathrm{ceil}(T*p(r)/C), \qquad 3.2(11)$$

and with p(r) of equation 2.1(4) yields

$$AC = \sum_{1}^{D} \mathrm{ceil}(1/r*\ln(D+g))*1/(r*\ln(D+g)).$$

This was calculated by Lowe [44], ~~who~~ also provides upper and lower bounds for AC.

It must be remembered that the inverted index is not the only source of disk accesses, since during the decoding operation one disk access is necessary for each term to be decoded. The quantities required for system evaluation are P for the disk storage utilization, AC for the number of disk accesses, and the compression ICR. To illustrate the preceding analysis figures for SD, ICR and SI are provided in Table 2 and are computed for different values of the database parameters. The formula used for T as a function of D is the one given by Heaps [29] for Chemical Title Tapes. It shows clearly the magnitude of the quantities involved in the analysis.

D	AW	T	H(bits)	L(bits)	ICR	SD(bits)*
10,000	5	$3.3\ 10^{6}$	9.54	$1.6\ 10^{8}$	.199	$5.3\ 10^{5}$
	6			$1.9\ 10^{8}$	.170	$6.4\ 10^{5}$
	7			$2.1\ 10^{8}$	.149	$7.5\ 10^{5}$
100,000	5	$3.3\ 10^{8}$	11.50	$1.6\ 10^{10}$	.240	$5.3\ 10^{6}$
	6			$1.9\ 10^{10}$	.205	$6.4\ 10^{6}$
	7			$2.1\ 10^{10}$	.180	$7.5\ 10^{6}$
1,000,000	5	$3.3\ 10^{10}$	13.41	$1.6\ 10^{12}$	.279	$5.3\ 10^{7}$
	6			$1.6\ 10^{12}$	.279	$5.3\ 10^{7}$
	7			$2.1\ 10^{12}$	.210	$7.5\ 10^{7}$

* NOTE: AD = 1.333*AW was used in the formula for SD.

NR	D	S(bits)	ST(bits)**
10,000	10,000	$4.3\ 10^{7}$	$4.3\ 10^{7}$
	100,000	$4.3\ 10^{9}$	$4.3\ 10^{9}$
	1,000,000	$4.3\ 10^{11}$	$4.3\ 10^{11}$
100,000	10,000	$5.3\ 10^{7}$	$5.3\ 10^{7}$
	100,000	$5.3\ 10^{9}$	$5.3\ 10^{9}$
	1,000,000	$5.3\ 10^{11}$	$5.3\ 10^{11}$
1,000,000	10,000	$6.3\ 10^{7}$	$6.3\ 10^{7}$
	100,000	$6.3\ 10^{9}$	$6.3\ 10^{9}$
	1,000,000	$6.3\ 10^{11}$	$6.3\ 10^{11}$

**NOTE: The difference in magnitude between SI and SD makes ST nearly independent of D.

TABLE 2

STORAGE REQUIREMENTS FOR RETROSPECTIVE SYSTEMS

3.3 Retrospective Systems Based on Fragments

The discussion in the previous section indicates that there are some disadvantages to the organization of a retrospective system as described. A brief summary of them is as follows:

A) Processing of the entire database is necessary before the dictionary size is known.

B) During coding the dictionary must be constantly updated and kept in an order suitable for locating the input words rapidly.

C) Coding and decoding require frequent disk acesses to the dictionary, because the dictionary is too large to be kept in core.

D) The disparate list length of the inverted index makes its organization into buckets clumsy and results in a lot of wasted disk space.

It seems that one solution to these problems is to adopt the proposal of Clare et al. [17] that the basic language and indexing elements be changed from words to fragments. A specific property, that of equifrequency, was imposed on the fragments to be used in order to overcome the above difficulties. Changing from words to equifrequent fragments would eliminate the disadvantages above, because

a) A small representative sample of the database should be sufficient to generate a fragment dictionary with the desired properties.

b) Since the dictionary is determined from a sample, before coding the entire database, it can be arranged in the order necessary for coding and no further updating is required.

c) The size of the dictionary can be held variable through adjustment of the frequency level. Thus it can be limited in size and could conceivably be stored in core.

d) The equifrequency of the selected indexing elements guarantees approximately uniform list length for the rows of the inverted index. With the proper choice of bucket capacity the unused disk space can be virtually eliminated.

However it should be realized that the fragment approach may also have some drawbacks. It is the purpose of the present study to investigate the feasibility of a fragment oriented system and report on the seriousness of the problems encountered. At this stage the following obvious questions arise:

1) Can a representative database sample be found and can at least approximate uniformity of the frequency distribution of dictionary fragments be obtained ?

2) How serious is the increase in complexity of the coding process when variable length strings are used ?

3) Are the storage requirements for a fragment system the same as for a word oriented system ?

4) Will the change of indexing elements result in the same

system performance, or will the number of false hits increase sharply and thus make the system impractical ?

These questions will be answered in the remaining chapters but, to keep direction and continuity in the investigation, concentration must center for the moment on the few parameters governing the system. Basically these parameters are the fragment equivalents of word oriented systems, the average fragment length AF, the number of dictionary elements ND, and the rank frequency distribution. Assuming that the fragments are approximately equifrequent, the average frequency F is sufficient to characterize that distribution, and the entropy is given by

$$HF = ld(ND) \quad . \qquad 3.3(1)$$

The number of fragments in the compressed database and in the inverted index is TK/AF with TK, the total number of characters defined as

$$TK = \begin{cases} TC & \text{for Word fragments} \\ TC+T & \text{for Text fragments} \end{cases}$$

since T is approximately the number of blanks in the database. This yields for the size of the compressed database, when fragments are used as language elements,

$$LCF = (TK/AF)*HF \quad . \qquad 3.3(2)$$

To facilitate comparison with other word compression schemes the equation for LCF may be written in the form

$$LCF = (HF * AW)/(AF*HW) * LC. \qquad 3.3(3)$$

If the factor that multiplies LC is greater than one it indicates that there is less compression than with word coding. With R defined by 3.2(6) the size of the inverted index SIF based on fragments as indexing elements is given by

$$SIF = (TK/AF) * R , \qquad 3.3(4)$$

and may be expressed in terms of the word index as

$$SIF = (AW/AF) * SI . \qquad 3.3(5)$$

For word fragments AF is less than AW and hence 3.3(5) indicates a larger index than for words. Only with text fragments does there exist a possibility of reducing the storage requirements below the value of SI. However the simple formula of 3.3(5) does not then apply because of the presence of embedded blanks.

The storage requirement for the fragment dictionary is

$$SDF = ND * AF * CR , \qquad 3.3(6)$$

which with the proper choice of ND and AF may be made considerably less than SD. At this point it must be remembered that F, ND and AF are connected by the relation

$$TK = ND * AF * F . \qquad 3.3(7)$$

A choice of any two parameters determines the value of the third. A natural choice for one parameter to be set in the design of the system is the average frequency F, which by proper choice can eliminate unused disk space. If the hardware operates with a sector size of S bits the ideal choice for the average frequency is

$$F = S/ R , \qquad 3.3(8)$$

which results in a memory utilization factor of zero and an average number of disk accesses equal to one.

The other choices of values of parameters are not easy to make, since they depend on the manner in which dictionary fragments are selected, and on the availability of system resources. The product ND*AF is fixed by the choice of F, and this determines the size of the dictionary. With enough disk space but only a small core memory, a small value for AF can be tolerated since the increase in SIF is not serious in this instance. However in the instance of limited disk space, AF should be made as large as possible. The relation between ND and AF is determined by the way the dictionary fragments are selected. This is an important step in the system design and must be made with great care. An algorithm for the selection of dictionary fragments is developed and described in the next chapter.

CHAPTER IV

SELECTION OF EQUIFREQUENT FRAGMENTS

The present chapter describes a heuristic method for the selection of fragments, and outlines the considerations and arguments which led to the development of the method. Experimental results will be used to justify this approach and to suggest equations which can be used to predict the critical parameters. Before proceeding with the description [illegible] should be mentioned that the length distribution of fragments may be calculated using a result from linguistics.

Assume that all fragments of length less than or equal to p are generated for a given sample of NR records with T tokens and D types. Let the number of fragment tokens be denoted by FT(p) for text fragments and FW(p) for word fragments. These sets are generated by moving a fixed size window across the record (word) and recording all the fragments displayed in the window. This is done for all records and window sizes up to p. The fragments can be sorted and compressed to give a frequency list of fragment types; an excerpt of this list is shown in Table 3. The set of fragments can also be generated in the same manner, or instead reference may be made to a word frequency list.

Assume the length of record j is specified by L(j). The number of fragment tokens of length up to p is given by:

$$FR = \sum_{i=1}^{NR} \sum_{j=1}^{p} (L(i) + 1 - j), \quad L(i) \geq j \qquad 4.0(1)$$

and, using the total number of characters TK as defined in section 3.3, the sum 4.0(1) reduces to

$$FR = p((TK+NR) - NR(p+1)/2) . \qquad 4.0(2)$$

If it is known that there are z(i) records of length i, then 4.0(2) can be rewritten as

$$FR = \sum_{k=1}^{kmax} \sum_{j=1}^{p} z(k)(k+1-j) \ , \ k \geq j$$

and the relative frequencies of fragments of length j are given by

$$rf(j) = \sum_{k=j}^{kmax} z(k)(k+1-j)/FR . \qquad 4.0(3)$$

For word fragments the values of z(k) are known from equation 2.1(6) and hence rf(j) may be calculated. For text fragments the average record length RA can be substituted for L(i) since p is usually much less than L(i). This yields for the relative frequencies of text fragments of length j:

$$rft(j) = (1 - j/(RA+1))*p^{-1} . \qquad 4.0(4)$$

This implies that the short fragments are more frequent than the longer ones. The number of fragment types however, cannot be calculated in terms of the parameters used above since it is a vocabulary dependent quantity.

The number of types is an important quantity to be considered in fragment selection, since it determines indirectly the size of the fragment dictionary. An inspection of the sets FT(p) and FW(p) shows that the combinatorial explosion during the fragmentation process leads to considerable duplication. The number of characters in FT(p) or FW(p) can be calculated as

$$FC = (p*(p+1)/2) * ((TK+NR) - NR(2p+1)/3) , \qquad 4.0(5)$$

which is almost p(p+1)/2 times the total number of characters in the database. However, this is not unexpected since there is considerable overlap among the fragments. Reduction of this over-redundant overlapping set into a manageable dictionary is the basic problem in the selection of dictionary fragments. It will be considered again when the relations among the fragments of FT(p) and FW(p) have been explored further.

4.1 <u>Relations between Fragments</u>

The fragmentation process, and the internal relations between the different fragments, can be explained by borrowing and adapting terminology that has been widely used for the description of graphs and trees.

A fragment is said to be of length k if it is a concatenation of k characters from the respective character set. It should be noted that for word fragments this set consists of alphabetic and numeric characters, but for text fragments ~~it~~ includes the "blank" or space character as well as all punctuation symbols. A set of fragments S(k) can be associated with each fragment of length k. The members of S(k) are all the proper subfragments contained within the k-letter fragment. The subfragment set S(k) may be structured using the inherent relations between a fragment and its subfragment. The explanation of the structure becomes easy with the following definitions:

__Node-Fragment__ : Each fragment of length k>1 can be considered as a node of a binary tree. The two branches which lead to nodes identified by fragments of length k-1 are obtained from the node-fragment by dropping either its first or last letter.

__Root-Fragment__ : The fragment which cannot be sprouted from another fragment in S(k) is called the root-fragment.

__Leaf-Fragment__ : Fragments of length one (single characters) cannot sprout further fragments and are the leafs or terminal nodes.

If fragments of the same length in S(k) are arranged on the same level, with the root fragment on level one, the generation of a unique complete binary tree is fully described, provided a rule for sprouting the two branches from a node fragment is specified.

__Sprouting Rule__ : Let the right(left) branch sprouted from a node fragment of length j (j>1) be the one which leads to the fragment of length j-1 that is obtained from the node fragment by peeling off the first(last) letter.

Application of this rule results in a complete binary tree of k levels whose nodes represent all elements of S(k). The 2**j nodes on level j can be numbered from left to right. This convention and the sprouting rule are sufficient for identifying each node in the tree for a given root fragment.

Inspection of the full tree reveals that some neighbouring node fragments on each level are identical. This duplication may be eliminated by merging these identical nodes level by level, but the resultant structure is no longer a tree. In terms of graph theory a gradable acyclic digraph is obtained [59]. This digraph is still closely related to the former tree structure, but at each level j there will be j nodes and the total number of nodes in the compressed tree is given by k(k+1)/2.

This type of graph appears in the theory of symmetric networks in switching theory [38] and in the social sciences [28]. It has been called "reproduction graph" by Ore [60] and "status graph" by Harary [28] when used to desribe the status of individuals within an organization. For this reason the compressed fragment tree that specifies all subfragments and their relations will be called an S-graph. If it is found necessary to indicate the height of the tree, which is equal to the length of the root fragment, it is referred to as a S(k)-graph. An example of a binary tree and the corresponding S-graph is shown in figure 3 for the word fragment "UNIT".

It can be seen from the figure that neighbouring fragments on the same level of an S-graph have common substrings. This overlap on level j of a S(k)-graph consists of k-j letters.

Using the described rules, S-graphs for all database fragments of length not greater than p may be constructed.

It is immediately obvious that the S-graphs for two distinct fragments F(1) and F(2) of length L(1) and L(2) need not to be independent of each other. Three cases of relationship between the two S-graphs S^1 and S^2 can be distinguished and are depicted in figure 4.

Total dependency: One of the S-graphs, say S^1 is a subgraph of S^2. This must imply that $L(1) \leq L(2)$ and hence fragment F(1) is completely embedded in F(2).

Partial dependency : Both S-graphs share a common subtree of say m levels. This indicates that a character string of length m is contained in both fragments F(1) and F(2). [illegible] F(1), this string is prefixed by L(1)-m letters, while L(2)-m characters are the suffix to the common substring in F(2).

Total independence: The two S-graphs have no common nodes, which implies that they have no letters in common.

The first and the last case are trivial and of no interest. In connection with the second case the question arises regarding the number of different S-graphs that share the same subtree. This can be restated as:

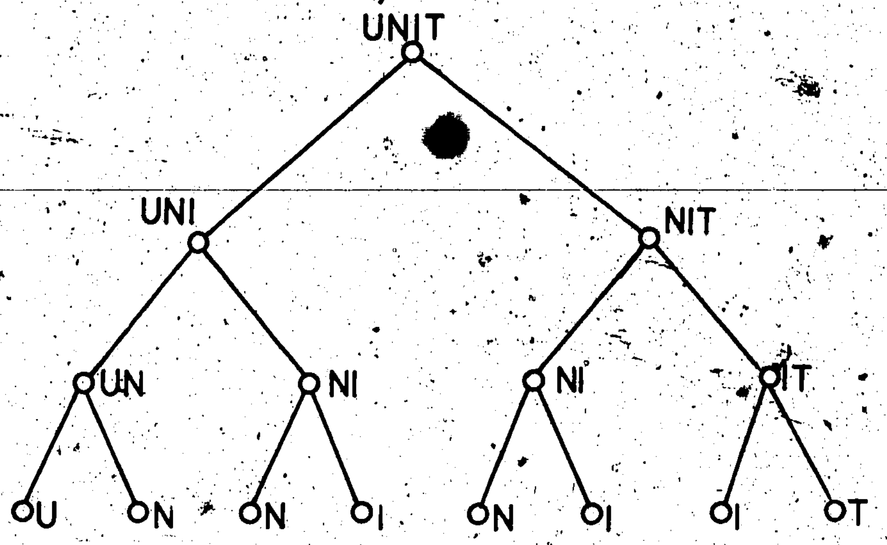
How many different fragments contain a specified substring ?

An answer is immediate for k-letter fragments that contain a common substring of k-1 characters. To construct a fragment of length k with a specified subfragment F' of length k-1 it is sufficient to add a single character from the character set. Any of the letters L can be concatenated

35	UNI	1	UNIC	1	UNICI
1	UNICIP	1	UNICIPA	1	UNICIPAL
1	UNIF	1	UNIFO	1	UNIFOR
1	UNIFORM	1	UNIG	1	UNIGU
1	UNIGUN	1	UNIGUND	1	UNIGUNDE
5	UNIO	5	UNION	1	UNIONS
2	UNIS	1	UNISM	1	UNIST
18	UNIT	16	UNITE	16	UNITED
2	UNITY	7	UNIV	7	UNIVE
7	UNIVER	7	UNIVERS	1	UNIVERSE
6	UNIVERSI	2	UNN	1	UNNE
1	UNNI	1	UNNIN	1	UNNING
1	UNNINGH	1	UNNINGHA	1	UNO
2	UNP	2	UNPO	2	UNPOW
2	UNPOWD	2	UNPOWDE	2	UNPOWDER
1	UNSELI	1	UNSE	1	UNSEL
1	UNSELI	1	UNSELIN	1	UNSELING

TABLE 3

SECTION OF FREQUENCY LIST FOR WORD FRAGMENTS



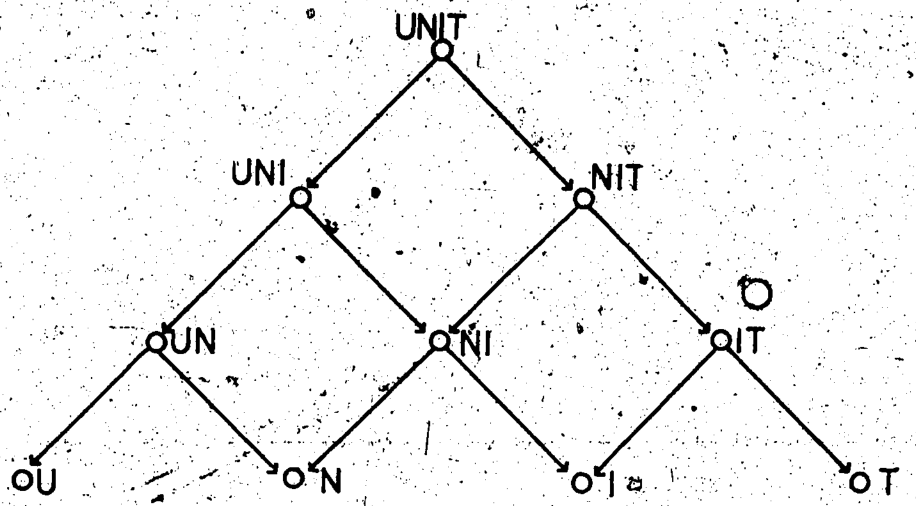
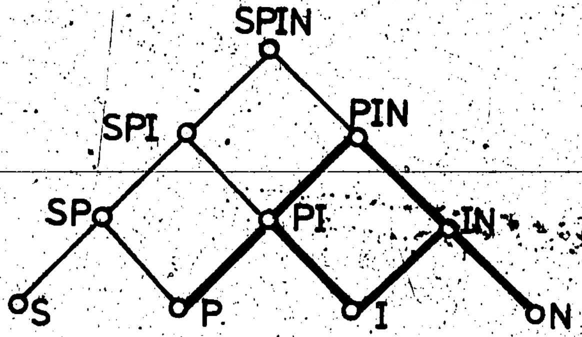
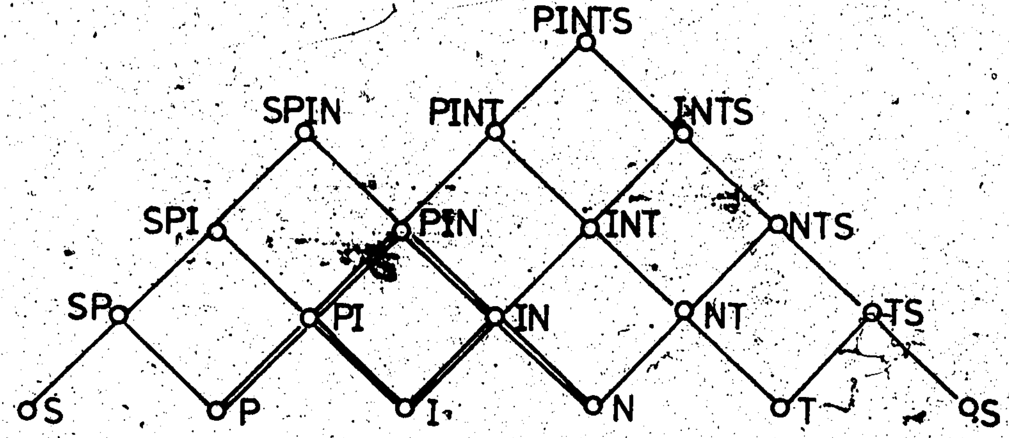


Figure 3

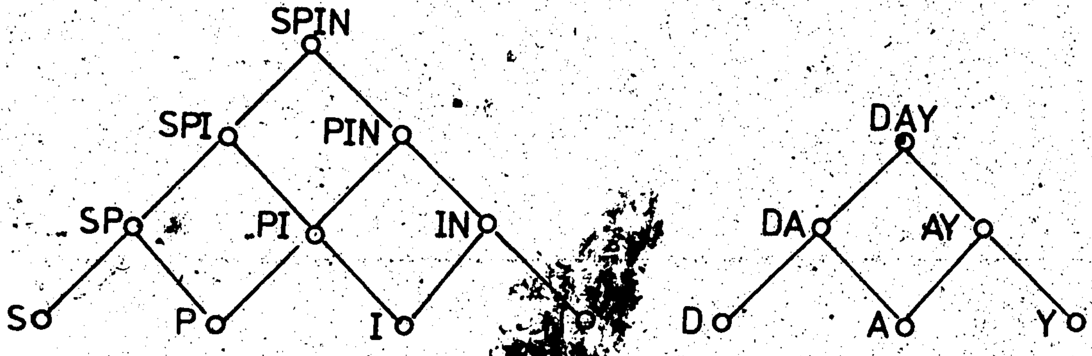
Binary Tree and S-Graph for Fragment "UNIT"



Total Dependence



Partial Dependence



Total Independence

Figure 4

The three Cases of Relationship between S-graphs

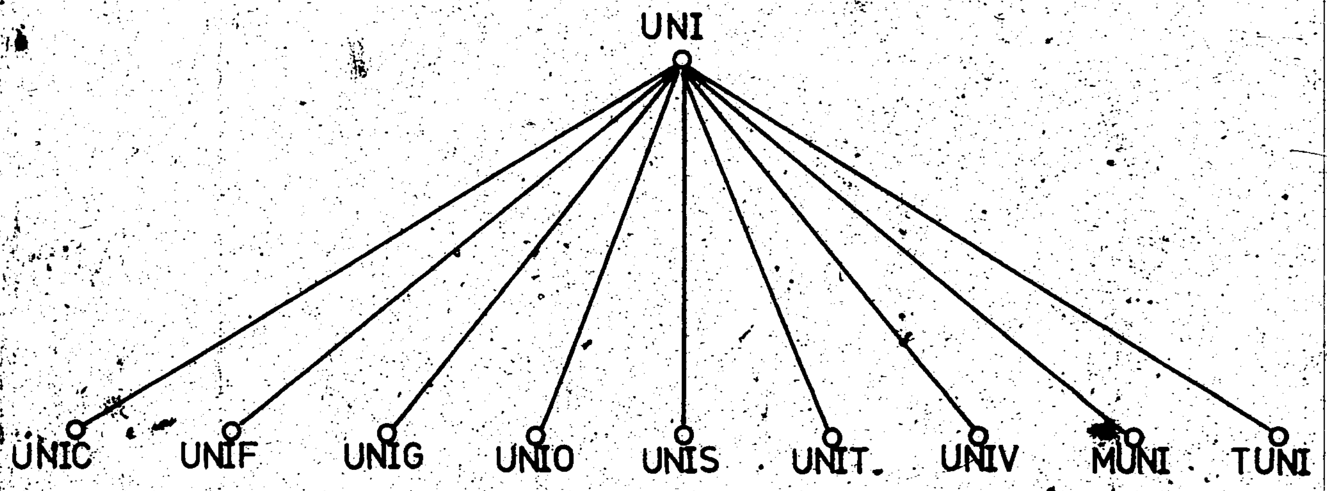


Figure 5

Relation between K - 1 and K letter Fragments with F' in two ways namely as L||F' or as F'||L. If the character set has r symbols there are therefore not more than 2*r different fragments of length k that contain F'.

This dependency can be shown by using F' as a root node of a new tree which sprouts all possible derivations of k-letter fragments from the F' node. From a knowledge of the vocabulary of the language used, it can be shown that not all 2*r combinations of F' and a letter form elements that actually occur in the language. Thus the number of branches that leave the root node associated with F' is smaller than 2*r. An example is provided by figure 5 which has root node F1 ="UNI" and all the branches for the sample text.

It is obvious that this concatenation process of letters and fragments is recursive and leads to the construction of a tree that shows all fragments. The depth of this tree is the maximum fragment length. This approach points to a method of storing all the fragments generated from a database sample. The following section will describe this in more detail.

4.2 Storage of Fragments

The tree described at the end of the previous section is capable of displaying all fragments occuring in the database sample but it has one serious disadvantage. Each fragment is stored twice, or more, and subtrees are duplicated since each k-letter fragment can be derived by prefixing or suffixing the k-1 letter fragment with a single letter. This duplication can be eliminated by applying only one of the two concatenation rules. The one selected in this study is the suffix rule. Only fragments derived from shorter ones by suffixing a single character are shown as branches from the proper node. This approach has been used previously by Briandais [8] and the storage and search properties of such a tree-organized system have been described by Scidmore and Weinberg [80].

The set of all database fragment tokens of length up to p contains many identical fragments. From the full set a smaller set F(p) of fragment types can be obtained by recording the fragment and its frequency of occurrence. In F(p) each element is therefore characterized by the fragment and its associated frequency. For the storage of F(p) the full tree with 2*r branches on each node is not needed since F(p) does not include all the (r**(p+1)-1)/(r-1) possible letter combinations. To store the fragments of F(p) it is sufficient to use only the branches needed. If F(p) is ordered by length as the primary key, and by alphabetic rank within each length group, the construction of the tree can follow well known methods. However, the associated frequency for each fragment must also be stored with each node. This data structure, which is actually a forest since the single letters form the root nodes of separate trees, plays an important role in the selection algorithm and will be called a "selection forest". Part of this forest is shown in figure 6 and will be referred to subsequently.

The storage of the selection forest does not cause any problems since it can follow well established procedures. It should be implemented with upward and downward pointers to permit a traversal of the tree in two directions. The unique concatenation rule makes it unnecessary to store the associated fragment at each node; the letter to be suffixed is sufficient. Each node fragment is then found by descending from the node to the root and recording all the letters encountered when visiting a node. This letter sequence forms the fragment associated with the node from which the descent started.

The selection forest possesses an important feature with regard to the frequencies associated with each node. The frequency of a root node is the sum of the frequencies of all the nodes in the filial set. This implies that the root frequency is always equal to or greater than that of any of the nodes in the filial set.

If the frequencies are also associated with the nodes of the S-graph a similar property is observed. The frequency of a lower level node is always equal to or greater than that of its root node on the higher level. The selection algorithm to be described relies heavily on these properties of the two structures.

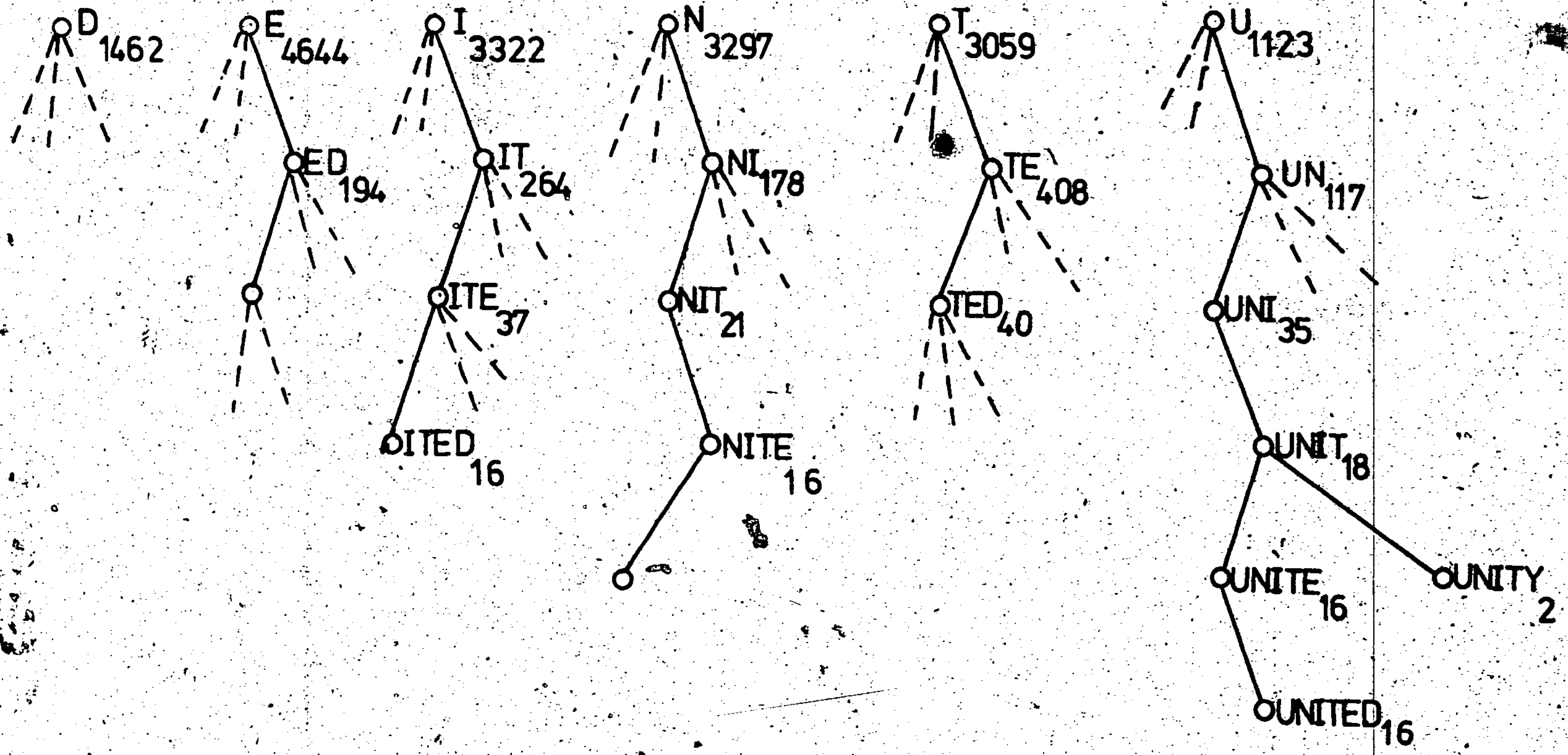


Figure 6

Section of Selection Forest

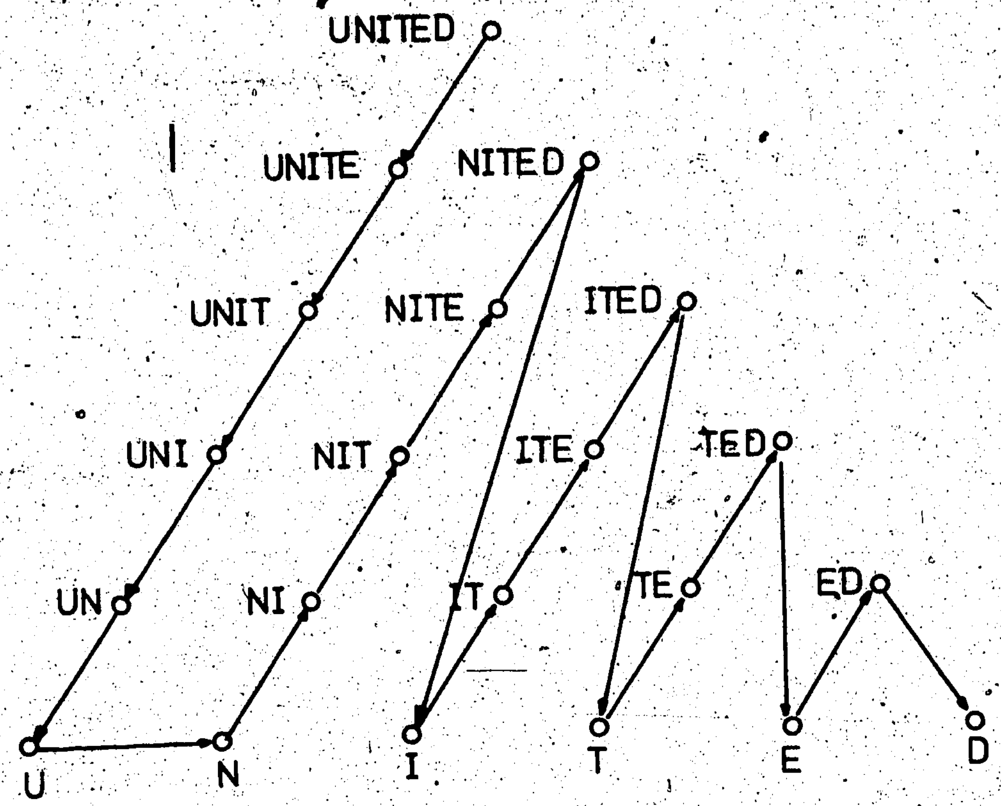


Figure 7

Sequence of Nodes to be traced out in the Selection Forest for Fragment "UNITED"

4.3 Selection Objectives

A summary of this section and a description of the resulting algorithm as described in section 4.4 can be found in [75], and a similar algorithm was outlined by Lynch et al. [46].

From the set F(p) of all different fragments a smaller subset, called the dictionary set DI, is to be selected. The over redundant set F(p) is unusable as an efficient dictionary because of the disparate frequency distribution. A reduction in the size of F(p) can be achieved by discarding certain fragment types. If a frequency threshold t is introduced two options are open:

1) Eliminate all fragments with an associated frequency above the threshold.
2) Eliminate all fragments with the frequency below the threshold t.

The first approach proposed by Clare et al. [17], and used by Lynch et al. [46], chops off the high end of the rank-frequency distribution of the fragments and hence leads to a more uniform distribution in the remaining set. However the reduction in the size of F(p) is not very significant, since the low end of the distribution accounts for most of the fragments in F(p). This can be seen from Booth's law which, although only partially applicable to fragments, still gives a good indication.

The method thus leads to significant reduction only if a very low threshold is selected. This is in sharp contrast to the requirements for effective compression of the database. It is preferable to introduce a code for a high frequency fragment rather than a low frequency one. The price to be paid in both cases is the necessity to store a copy of the fragment to be coded on expensive random access storage.

Assuming that equifrequency of the elements is used for coding, the number of characters in the dictionary DI is given as TD by the formula:

$$TD = TK/F , \qquad 4.3(1)$$

with F denoting the average frequency of occurrence. For $F \geq t$ this number TD is clearly less than for $F < t$. It therefore seems desirable to compress those fragments whose frequencies exceed the threshold and to neglect all those whose frequencies lie below it. Such an approach to compression with fragments has been selected in this study.

It must be pointed out, that the proposed approach could discard certain strings which occur in the database with low frequency and yet whose inclusion in a dictionary is an absolute necessity in order to ensure a complete representation of the database. The dictionary DI must be made complete in the sense that the whole database can be represented by concatenation of elements of DI. This can be achieved by including all members of the character set in the dictionary regardless of their respective frequencies of occurrence.

The choice of the threshold value, and the subsequent elimination of fragments with frequencies below it, reduces F(p) to a set with Q(t) elements of which some are still redundant. Many fragments of the reduced set have overlap, i.e. have common substrings, a feature which ideally should be avoided. Elimination of overlapping fragments from the set is a necessity and has the advantage of further reducing its size. When performing the reduction it must be kept in mind that the equifrequency property is the most important feature. The efficiency E of equation 2.3(3) is a measure of equifrequency and to improve on this property during selection it is desirable to maximize E.

A further aspect to be considered are the implications of equations 3.3(3) and 3.3(5) which involve the average fragment length AF. Since a small value of AF implies a relatively large demand on storage space it is imperative to keep AF as large as possible in order to keep the storage costs down.

Dictionary fragments play a dual role, they are used as compression and indexing elements. If all single characters are included in the dictionary in order to guarantee completeness, it is clearly undesirable to have rows of the inverted index corresponding to single characters. This could cause an excessive number of false hits during retrieval. If single characters are omitted from the inverted index then it becomes impossible to retrieve words which are concatenations of single characters and do not contain at least a bigram or a longer fragment. Such a deterioration in recall should be minimized and can be achieved by maximizing the number of words which contain at least one bigram or some longer fragment.

The desirable properties to be taken into account during design of the selection algorithm can now be summarized as follows in which the items are listed in decreasing order of importance.

A) The set of dictionary fragments must be complete.

B) The dictionary fragments should be chosen so that their frequencies of occurrence lead to a maximization of E. This condition is equivalent to requiring the fragments to be equifrequent.

C) The set of fragments should be chosen to maximize the average fragment length.

D) The database should contain very few words whose representation requires concatenation of single characters only.

E) The set of dictionary fragments should not be over-redundant in the sense of having unnecessary overlap between its members. Also it should not be possible to choose a smaller set that satisfies requirements A through D.

The extent to which these objectives have been realized in the actual implementation of the algorithm is described in the next section.

4.4 Selection Algorithm

The selection algorithm may be described in terms of the S-graph and the selection forest. It can be interpreted as a tracing of the S-graph in the selection forest, and manipulation of the node frequencies. From the discussion in the previous section it is obvious that any algorithm must consider the longest fragments first in order to maximize the average fragment length. If fragment F, whose frequency f is greater than the threshold t, is to be a candidate for inclusion in the dictionary then its removal from the subset of F(p) will affect some other members of the set. These are all the subfragments of F that appear together with their frequencies, in the S-graph of F. These subfragments can easily be generated from F since the construction rules for the S-graph are simple. On selection of F the frequencies of all subfragments must be reduced by f since their occurrence is decreased when the string F is included in the dictionary and hence removed from the portion of the database that is still to be represented. Such a frequency reduction is normally possible, since the frequencies of the nodes below the root in the S-graph are all greater than or equal to that of the root.

The S-graph is only a temporary construction used to find all the subfragments, and hence the adjustment of frequencies must be done in the selection forest which is the permanent structure for the storage of all fragments and their frequencies. The reduction of node frequencies is achieved by tracing out the S-graph in the forest; this is a simple operation when all the pointers are provided. A suitable node sequence for the word fragment "UNITED" is shown in figure 7 and the corresponding relevant portions of the selection forest are shown in figure 6.

When proceeding with the selection as described, and considering other fragments for inclusion in the dictionary whenever their frequency exceeds the threshold, the scheme will eventually fail. The adjustment of node frequencies during removal of previously selected elements will cause two or more nodes in the S-graph to have a frequency smaller than that of the root fragment. This is not critical; it indicates that part of the fragment candidate is already covered by some fragment of the dictionary. This fragment should be discarded, since the elimination of overlapping fragments is one of the main goals. The situation described above provides a very simple criterion for throwing out overlapping fragments. A simple test is sufficient and may be made when tracing out the S-graph in the selection forest.

After the above process has been performed for all p-letter fragments, those of length p-1 can be tested and the scheme can be repeated until single letters are reached. The remaining frequencies at the root nodes of the forest are the predicted occurrences of single letters in the representation of the coded database. The ideal algorithm would reduce these frequencies to zero and thus satisfy requirement D of the previous section.

A mathematical formulation for this process is possible but difficult and cumbersome. It can be treated as a nonlinear integer programming problem for which a solution may be found by enumeration of all possible sequences of nodes. This has not been attempted for two reasons.

Firstly, the combinatorial explosion during enumeration leads to excessive demands on computer time because of the large number of nodes involved. Secondly it was felt, that the statistical fluctuations in the database sample would result in a very sample dependent solution. Instead, it was decided to investigate the use of a heuristic algorithm in the hope that the selected set might prove almost as good as the best possible one that could be obtained by enumeration. The criterion by which to measure the quality of the selected dictionary was chosen as the efficiency E.

The algorithm operates under the following premises. The set F(p) of fragment types is available and with each fragment there are associated two extra quantities. These are the frequency of occurrence in the database, and the number of different records (words) in which the fragment is found. This last quantity is easy to extract during preparation of the frequency list. The selection forest is constructed for F(p), and to each node a fragment and its two associated quantities are attached. However only fragments whose frequency exceeds the threshold are used when building the forest. This saves storage and computing time. The steps of the algorithm are as outlined below:

ALGORITHM :

Step 1) Set L=p

Step 2) From all nodes on level L of the selection forest discard those with a frequency less than the threshold t. Order all remaining N(L) fragments of length L into a sequence S by performing a sort. The primary sort is on the node frequency in ascending order, the secondary sort in descending order on the number of different records containing the fragment .

Step 3) Set j = 0

Step 4) Set j = j + 1

Step 5) Locate all nodes of the S-graph of the j-th fragment of S in the selection forest. Let f(j) denote the frequency of this j-th fragment .

Step 6) If any of the nodes have a frequency less than f(j), go to Step 8.

Step 7) Include the j-th fragment of S in the fragment dictionary. Reduce the frequencies of all nodes involved by f(j).

Step 8) If j≤N(L) go to Step 4.

Step 9) Set L = L - 1.

Step 10) If L≥2 go to Step 2.

Step 11) Stop. Selection complete, include single characters for completeness.

It is evident that this algorithm attempts to satisfy the five requirements listed at the end of section 4.3. The first one is taken care of by including the character set in the dictionary, while the second is satisfied by inclusion of the sort in step 2, which put special emphasis on fragments with a frequency close to the threshold. Requirement C is satisfied by starting with the longest fragments first. Requirement D is met by including the secondary sort in step two. Step six is included because of requirement E.

The algorithm as described can be used for word, as well as for text, fragments. This algorithm was implemented and its performance recorded. Special attention was devoted to the equifrequency property and the average fragment length. The results are presented in the next section.

4.5 Experimental Results

To test the proposed theory and the performance of the selection algorithm, it was decided to use a sample of a database that is in frequent use. It is obvious that the sample could not be of the same magnitude as a database used by an actual retrospective system, because of the large expense in computer time required for the manipulation of the large amount of real data.

The database chosen for the experiment consisted of the MARC tapes as issued by the Library of Congress [35]. Two samples, different in size and structure, were used. The first was an issue of the MARC tapes for 1969. It was available in the form of complete records consisting of author, title, and subject heading fields. This sample is characterized by Table 4 and the distribution of word and record lengths is shown in figure 8.

The second, much larger, sample was available as a word frequency list. This list was compiled from author, title and subject heading fields for all MARC tapes issued in 1969. The sample characteristics are specified in Table 5 and the word length distribution is shown in figure 9. The first of the two samples provided the base for experiments with word and text fragments, but the second permitted only experiments with word fragments.

The experiment was started by using the first database sample to generate the set FT(10) of all text fragments of length up to 10 and also the set FW(8) of word fragments of length up to 8. These sets were compressed and a frequency list prepared. The above values for the maximum length were chosen because the longer the fragments are, the smaller their frequency of occurrence. This is confirmed by Table 6. Since fragments with a frequency below the threshold will be eliminated by the selection algorithm, only very few long fragments would be included in the dictionary. The small improvement in compression gained by including the few long ones whose frequencies exceed the threshold does not justify the computational effort that would be involved. The maximum length for the fragments was chosen so that only 25% of the fragment types have a length greater than the maximum. Table 6 shows the distribution of fragment types and tokens for the word and text fragments of sample 1. Figure 10 shows the distribution of the average rank and frequency for certain length groups. It may be noted from Table 6 and figure 10 that allowing the fragments to extend across word boundaries merely increases the number of tokens (and types) in each length group and has relatively little effect on the form of the distribution. Since both the average rank and f(r) in figure 10 are plotted on logarithmic scales, a distribution according to the Zipf law would result in a straight line. Such is clearly not the case, although the curves for the longer fragments are more linear than those for shorter ones.

From sample two only word fragments could be generated and the same maximum length as for sample one was chosen. The distribution of types and tokens is displayed in Table 7 and the rank-frequency distribution for the same length groups as for sample one is given in figure 11. The regularity and similarity for the distributions makes one suspect that it should be possible to formulate a derivation based on statistical properties of letter associations. This aspect was not investigated since it was felt to be beyond the scope of this thesis.

MARC TAPE ISSUE 1 1969

608 Author Fields

495 Title Fields

493 Subject Headings

Word Length	Types	Tokens
1	36	538
2	57	619
3	103	913
4	311	623
5	403	796
6	463	1009
7	461	908
8	334	693
9	269	515
10	185	341
11	124	222
12	71	136
13	44	64
14	21	21
15	8	28
17	1	3
Total	2,891	[illegible]

13 Special Characters with a combined Frequency of [illegible]

Total number of characters (incl. blanks) 51,04[illegible]

Number of Blanks 5,808

Average Word Length 5.72

Average Entropy 8.68

Efficiency E .882

Repeat Rate $59.95\ 10^{-4}$

TABLE 4

CHARACTERISTICS OF SAMPLE 1

MARC TAPES: ALL ISSUES OF 1969

Word Length	Types	Tokens
2	351	47056
3	1050	63835
4	2659	36598
5	3441	45729
6	4374	46276
7	4650	50579
8	4544	44465
9	3842	31158
10	3026	25995
11	2003	15749
12	1258	10013
13	772	4630
14	441	3040
15	250	919
16	132	280
17	69	[illegible]37
18	33	78
19	21	43
20	7	25
21	5	10
22	2	9
23	1	1
24	1	1
25	7	7
Total	32,739	426,737

Total number of characters	2,610,245
Average Word Length	6.12
Average Entropy	11.06
Efficiency E_r	.737
Repeat Rate	$95.82 \cdot 10^{-4}$

TABLE 5

CHARACTERISTICS OF SAMPLE 2

TEXT FRAGMENTS

Length	Types	Tokens
1	50	51057
2	890	49453
3	5020	47859
4	13094	46265
5	19787	44671
6	24047	43080
7	26960	41493
8	28747	39910
9	29642	38334
10	29965	36767
Total	178,202	438,879

WORD FRAGMENTS

Length	Types	Tokens
1	49	45240
2	523	35067
3	2878	28187
4	5900	21926
5	6091	21926
6	4932	11853
7	3567	7924
8	2390	5004
Total	26,330	171,779

TABLE 6

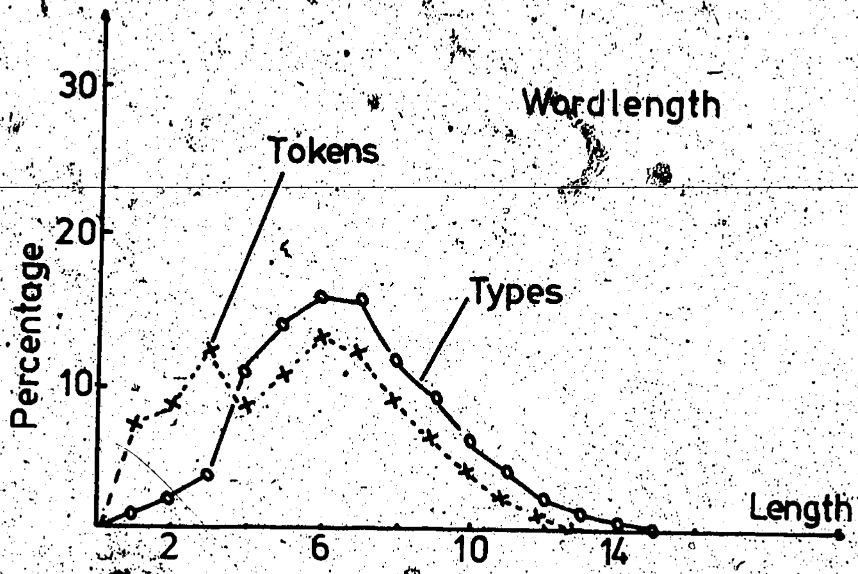
FRAGMENT TYPES AND TOKENS FOR SAMPLE

WORD FRAGMENTS

Length	Types	Tokens
2	865	2610245
3	7533	1756685
4	29097	1377018
5	50594	1061182
6	53878	781944
7	46023	548435
8	35092	361202
Total	223,082	8,498,711

TABLE 7

DISTRIBUTION OF FRAGMENT TYPES AND TOKENS FOR SAMPLE 2



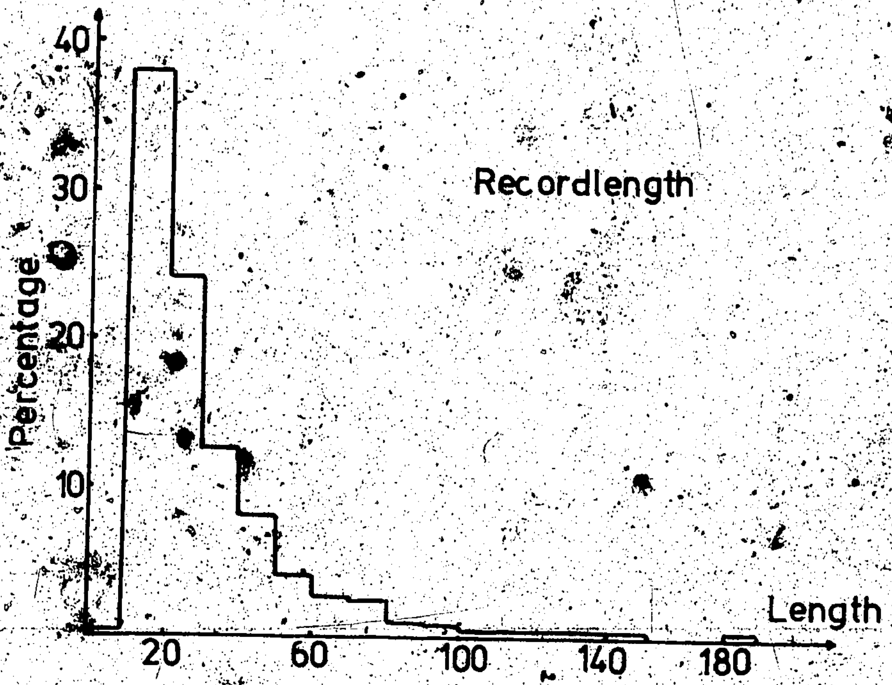


Figure 8

Distribution of Word - and Recordlength for Sample

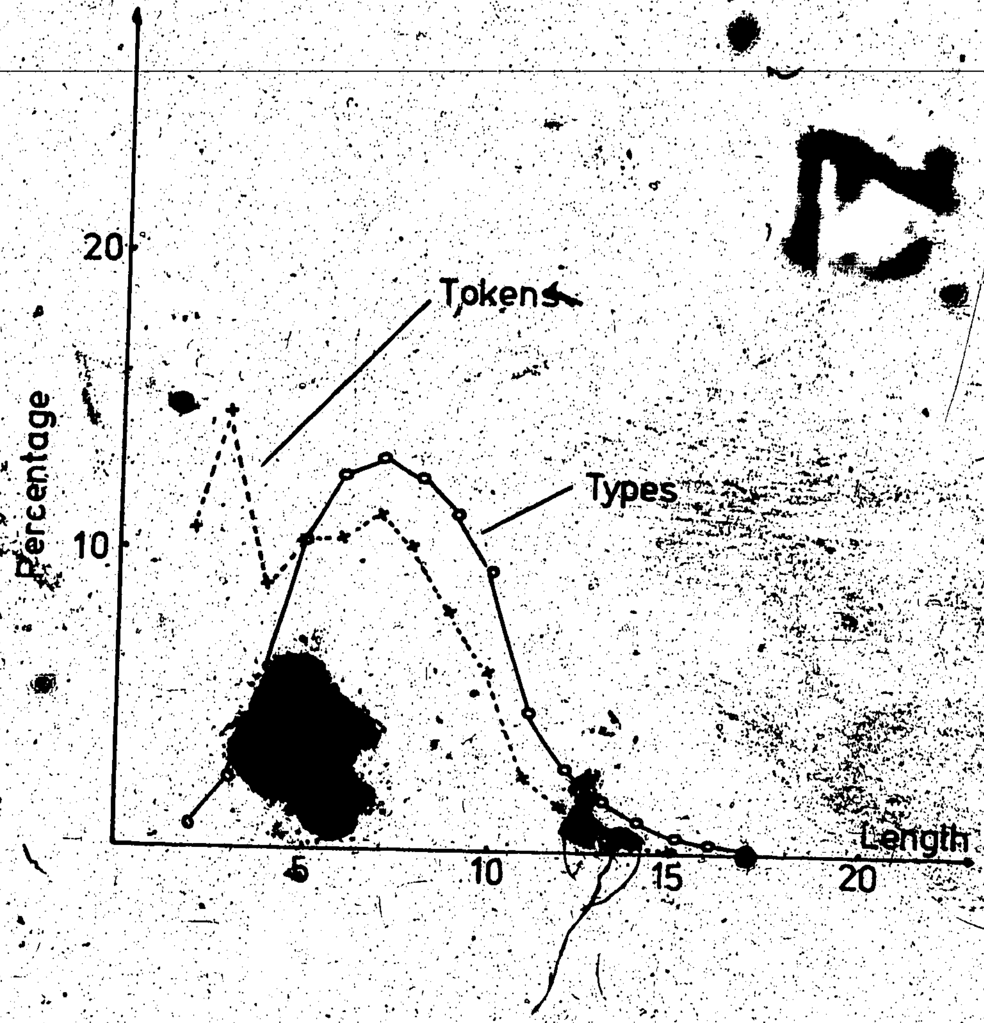


Figure 9

Wordlength distribution for Sample 2

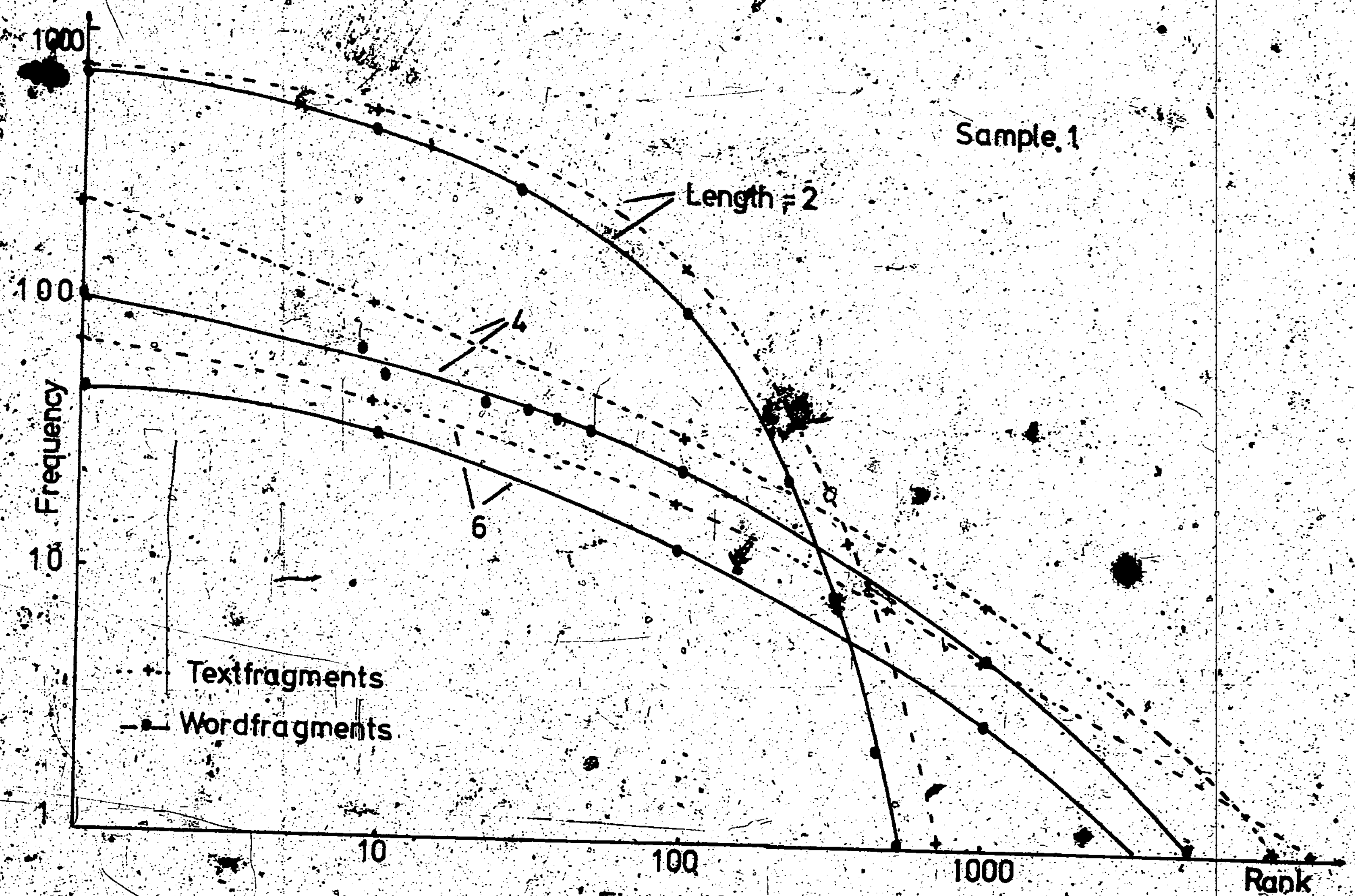


Figure 10

Rank - Frequency Distribution f[illegible] Fragments

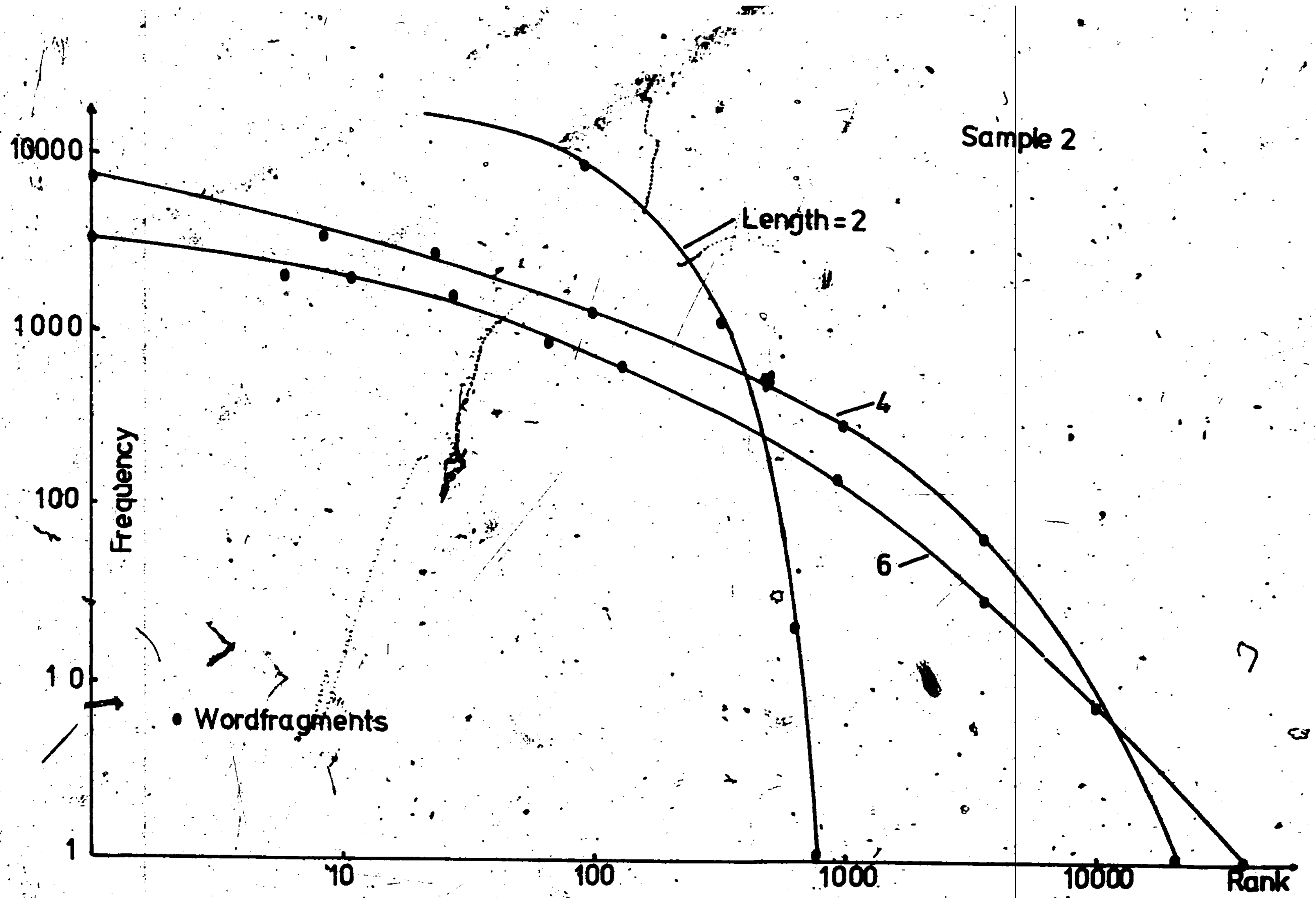


Figure 11
Rank - Frequency Distribution for Fragments

The selection forest must be stored before the algorithm can be applied and, since this algorithm involves only the fragments with a frequency above the threshold, all other fragments were eliminated. This not only decreases storage requirements but also the time to construct the forest, since only relevant nodes must be handled. The algorithm was implemented by a FORTRAN program that follows the description in section 4.4. The node sequence traced out in the selection forest follows the pattern of the example of figure 7.

The only parameter input to the selection algorithm is the threshold value t and it was decided to study the influence of t on the system and its operation. For each sample four different values of t were chosen. Absolute threshold frequencies for sample one were 5,10,15,20 and for sample two were 336,672,1008,1344. The reason for choosing the rather different values for the thresholds of the two samples was the sample size. The threshold values were selected to yield the same relative threshold frequencies in the set of fragment tokens for each sample.

Let Q(t) be the number of fragments that remain after elimination of those below the threshold (equivalent to the size of the selection forest) and let ND(t) be the number of dictionary fragments chosen by the selection algorithm. From section 3.3 it is known that the critical parameters that govern performance are the average fragment length, the size of the dictionary ND, the average dictionary entropy, and the efficiency E. The values of these main parameters that result from application of the selection algorithm are listed in Table 8. Two sets of figures are provided for each parameter, one set annotated by a quote. The difference between the two sets lies in the manner of calculation. Parameters AF, HF, E, F were calculated with all dictionary fragments. In contrast AF', HF', E', F' were calculated only with the set of index fragments which excludes the single letters. The first set of parameters is the one to be used in calculations that involve the compressed database, while the second set characterizes the inverted index and should be used in any formulas that relate to the index.

The appendix lists two sets of dictionary fragments selected by the algorithm. Each fragment is given together with each theoretical frequency of occurrence. The first set represents a dictionary of text fragmentsfor a threshold of 10, while the second is a word fragment dictionary for sample two and a threshold of 672. The set of dictionary text fragments shows considerable sample dependency, but this is less evident in the second set owing to the larger sample size.

When plotting the parameters of Table 8 against the threshold on log-log paper, as in figures 12 through 15, it is interesting and perhaps surprising, to see that they result in straight lines. These straight lines imply simple relations of the form:

Q(t) = Q0*t**-q 4.5(1)

ND(t) = N0*t**[illegible] 4.5(2)

[illegible]F(t) = A0*t**[illegible] 4.5(3)

[illegible](t) = H0*t**h 4.5(4)

[illegible](t) = F0*t**[illegible] 4.5(5)

for the size o[illegible] the sel[illegible] forest and the dictionary, as well as for [illegible]age [illegible] length, entropy HF, and average [illegible] the values of the constants in 4.5(1) - 4.5(5) were [illegible] by a least squares fit and are summarized in Table 10. [illegible] is fairly close agreement between the constants for [illegible] fragments of the two samples. It therefore appears that the exponents are independent of sample size. The validity of equations 4.5(1) -4.5(5) is [illegible]prising, especially with respect to sample two, when the wide range of the threshold values is considered. It is likely that these equations can be explained theoretically if a derivation for the distribution of fragments within each length group is found. This has not been attempted, since the objective was the investigation of the practical aspects of a fragment oriented retrieval system.

At this point it is also appropriate to compare the predicted sizes of the compressed database and the inverted index with the observed values. Table 9 summarizes the data and the results are in good agreement with the theory considering the unsophisticated approach taken in chapter three. The increase in the size of the compressed database can be attributed mainly to the difference between the entropy as the minimum average codelength and the actual codeword which must consist of an integral number of bits. The inverted index has fewer entries than predicted; this is due to the fact that the derivation of equation 3.[illegible] assumes that only index fragments are contained in the database. The number of single characters S(t) that should be subtracted from TK in the calculation of SIF is shown as S(t) in Table 8 and it accounts for the difference.

Another quantity to be examined is the expected improvement in storage utilization P and the number of disk accesses AC. The values of P and AC as functions of the bucket size are plotted in figures 16 and 17 for text fragments. Figures 18 and 19 display the same quantities for word fragments. For contrast, the values of P and AC for a word organized system are also shown. These plots confirm that the expected advantages of fragments can be realized and represent a substantial improvement over the use of words. Text fragments are slightly better than word fragments with regard to the average number of disk accesses, but the opposite is true for the storage utilization factor.

In both cases high thresholds lead to a larger number of disk accesses than do low ones, but the storage utilization is better for high thresholds. The choice of a threshold for the system (based on arguments related to the index) must consider these conflicting facts. This conflict of which threshold to choose is enforced by the fact that better compression is [illegible] threshold. However, the question of which threshold [illegible] for good overall system performance cannot [illegible] at this time since not all aspects have been investigated. This will be done in successive chapters. The main result of the present chapter is the discovery of the empirical [illegible] through 4.5(5) which allow the prediction of the [illegible] system parameters. This can be of great aid to the designer of a fragment oriented retrieval system.

	SAMPLE 1											
	TEXT FRAGMENTS				WORD FRAGMENTS				WORD FRAGMENTS			
THRES	5	10	15	20	5	10	15	20	[illegible]	672	1008	1344
Q(t)	7595	3819	2463	1657	4500	2107	1363	996	[illegible]	[illegible]	1070	743
ND(t)	1603	936	703	552	1387	824	[illegible]	490	1008	603	444	364
AF(t)	4.61	3.76	3.37	3.12	3.11	2.74	2.59	2.47	3.55	3.06	2.80	[illegible]
HF(t)	10.07	9.51	9.18	[illegible]	9.10	8.69	8.38	8.19	9.30	8.62	8.24	8.08
E(t)	.961	.967	.971	.965	.872	.897	.907	.917	.932	.934	.935	[illegible]
F(t)	7	14	22	30	12	23	33	42	731	1418	[illegible]	[illegible]
AF'(t)	5.02	4.28	3.82	3.61	4.01	3.50	3.23	3.07	3.91	3.50	[illegible]	[illegible]
HF'(t)	10.01	9.73	9.29	8.92	10.20	9.50	8.98	8.66	9.49	8.81	8.40	8.10
E'(t)	.993	.993	.994	.994	.982	.990	.986	.986	.957	.964	.966	[illegible]
F'(t)	7	13	19	26	[illegible]	15	23	29	664	1247	[illegible]	[illegible]
S(t)	1985	2127	2441	3103	8452	10367	[illegible]	11755	[illegible]	[illegible]	[illegible]	[illegible]

TABLE [illegible]

Results from Selection Algorithm

TEXT FRAGMENTS

Threshold	5	10	15	20
Database length predicted*	102,000	112,000	126,000	144,000
Database length observed	123,410	135,320	151,350	163,500
Number of Index Entries predicted*	10,100	11,700	13,150	13,900
Number of Index Entries observed	9,851	11,405	12,694	13,247

WORD FRAGMENTS

Database length predicted*	132,100	112,000	126,000	144,000
Database length observed**	130,941	135,320	151,350	163,500
Number of Index Entries predicted*	11,200	12,900	13,150	13,900
Number of Index Entries observed	10.195	11.496	12,694	13,427

* Parameters predicted by equation 3.3(2) and 3.3(4)

** These figures do not include overhead necessary with word fragments

TABLE 9

COMPARISON OF PREDICTED AND OBSERVED VALUES FOR THE SIZE OF THE INDEX AND THE COMPRESSED DATABASE OF SAMPLE 1

	SAMPLE 1		SAMPLE 2
	Text Frag.	Word Frag.	Word Frag.
ln Q0	10.91	10.14	14.91
q	1.145	1.082	1.151
ln N0	8.60	8.44	11.15
n	.762	.751	.730
ln A0	1.946	1.406	2.491
a	.270	.166	.211
ln H0	2.45	2.33	2.84
h	.087	.076	.105
ln F	.256	1.03	1.07
f	1.05	.903	.950

TABLE 10

CONSTANTS FOR EQUATIONS 4.5(1) - 4.5(5)

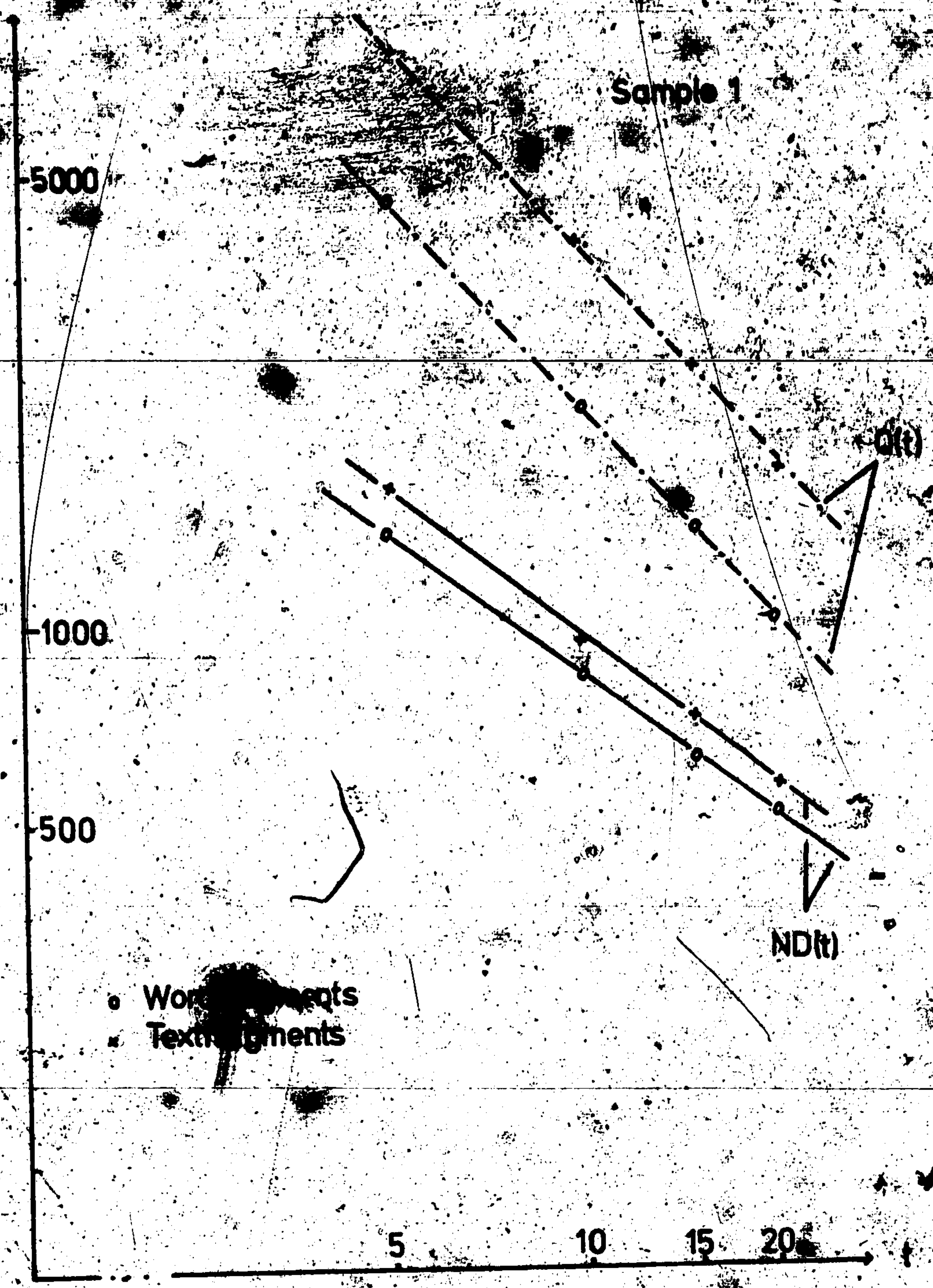


Figure 12

Size of Selection Forest and Dictionary versus the Threshold

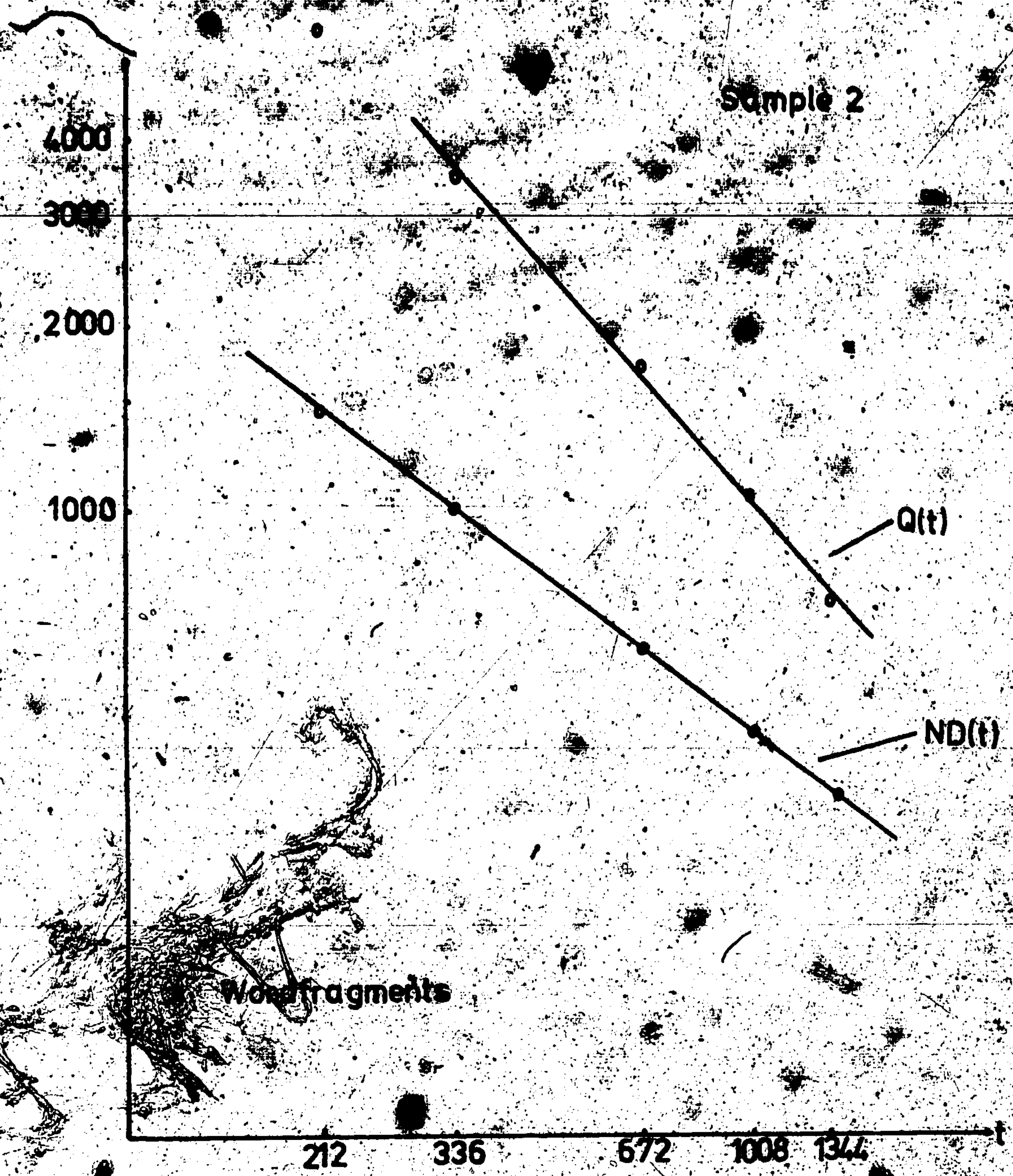
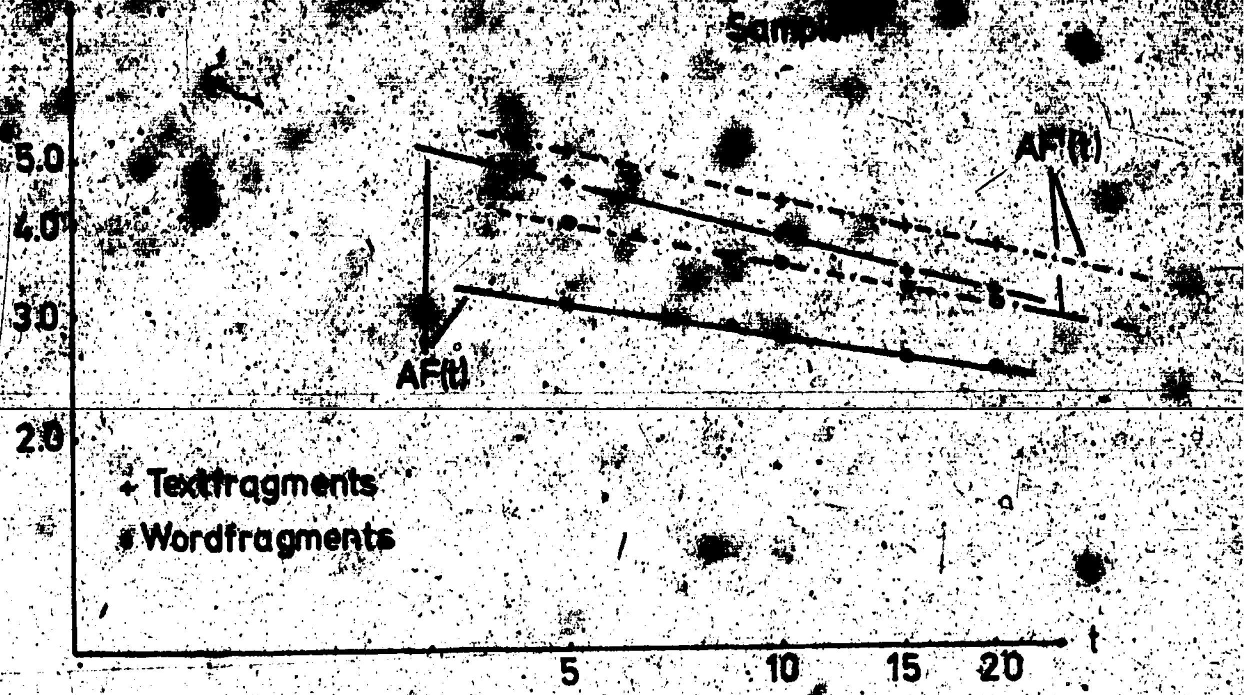


Figure 13

Size of Selectionforest and Dictionary versus the Threshold



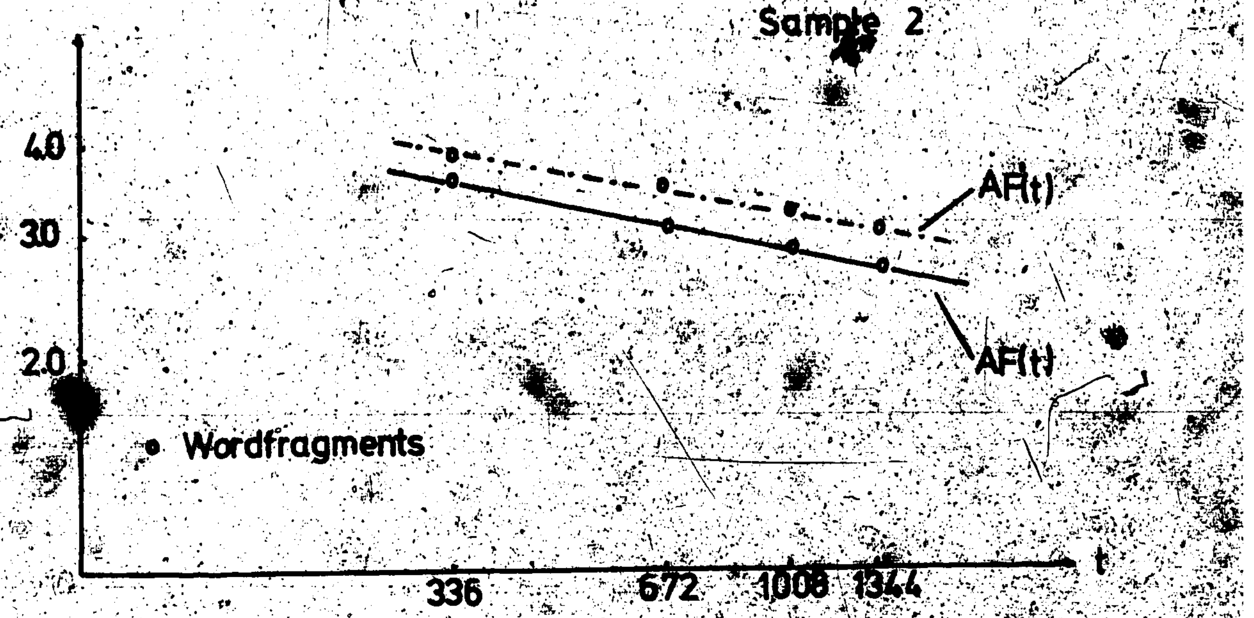
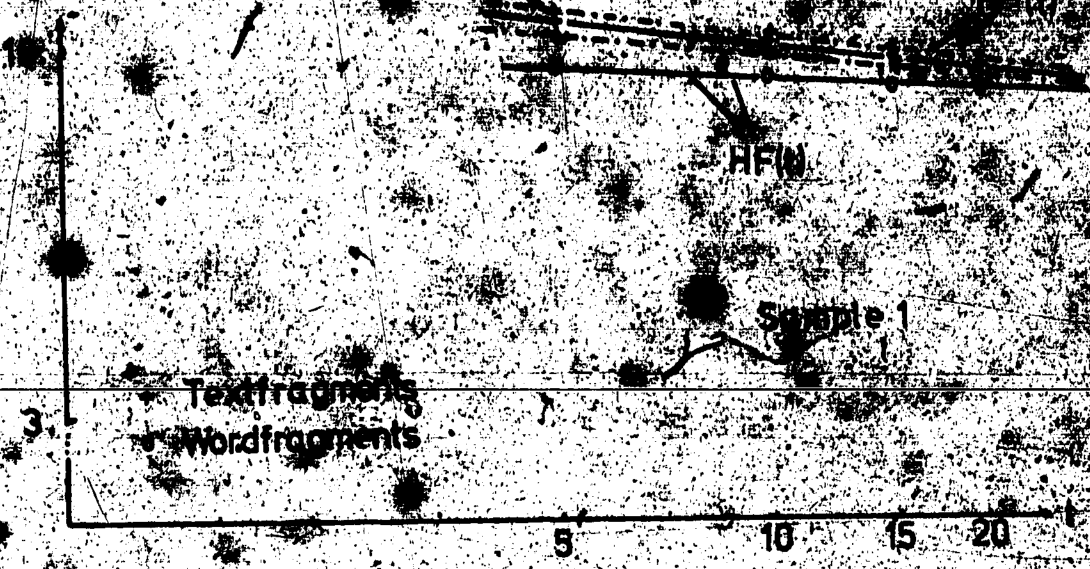


Figure 14

Average Fragmentlength versus the Threshold



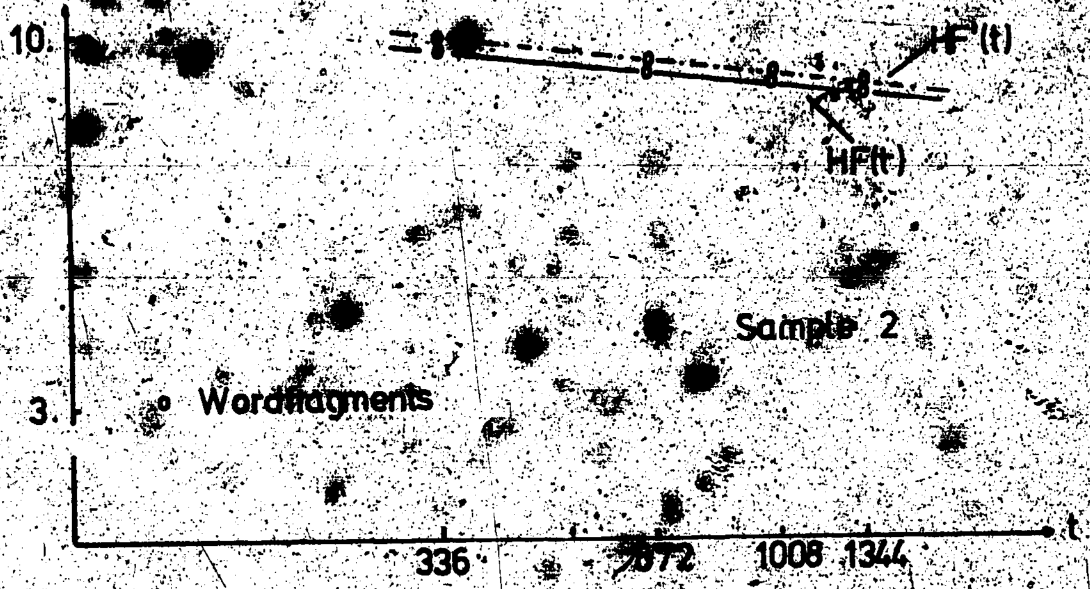


Figure 15

Average Entropy versus the Threshold

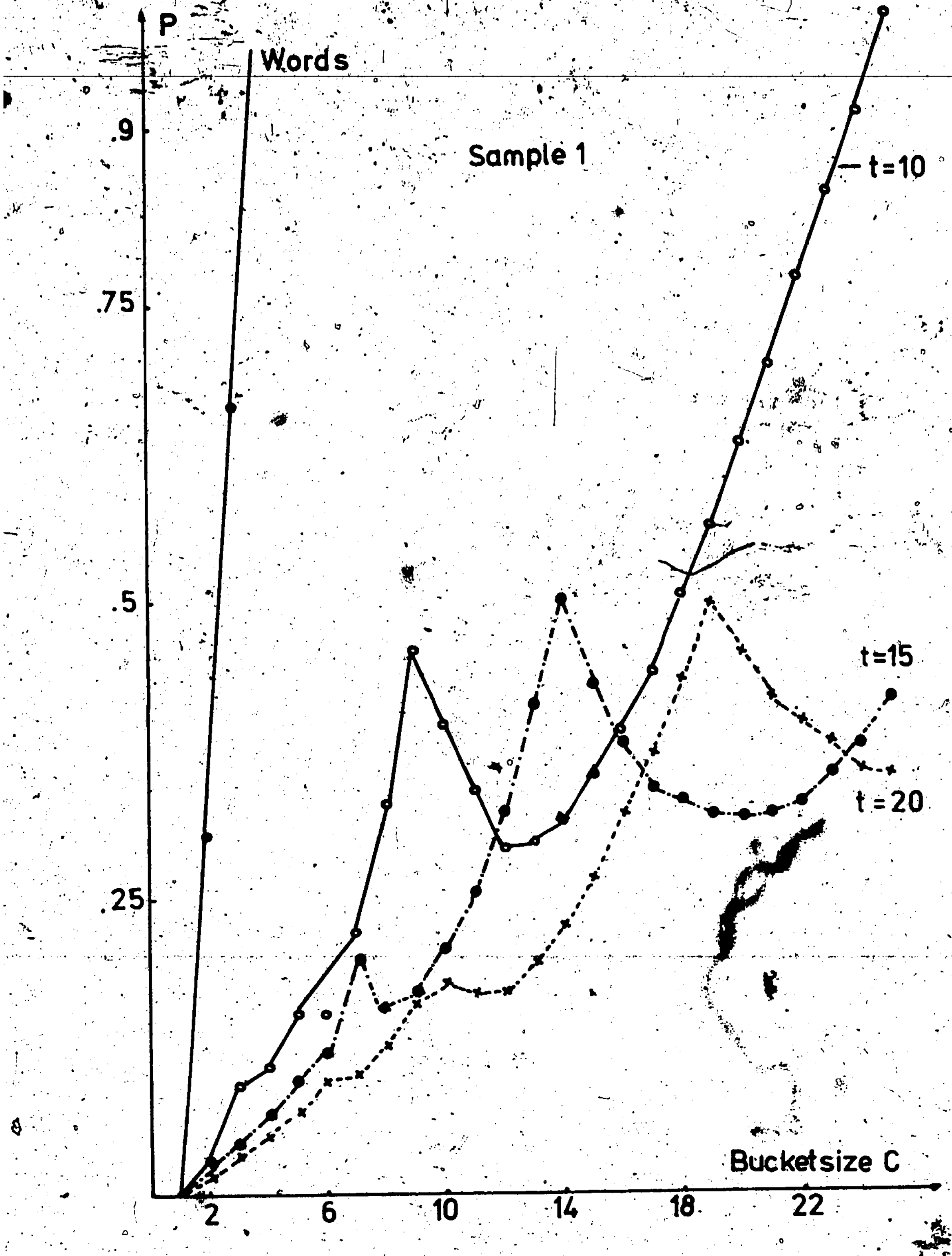


Figure 16

Memory Utilization Factor for Textfragment Dictionaries

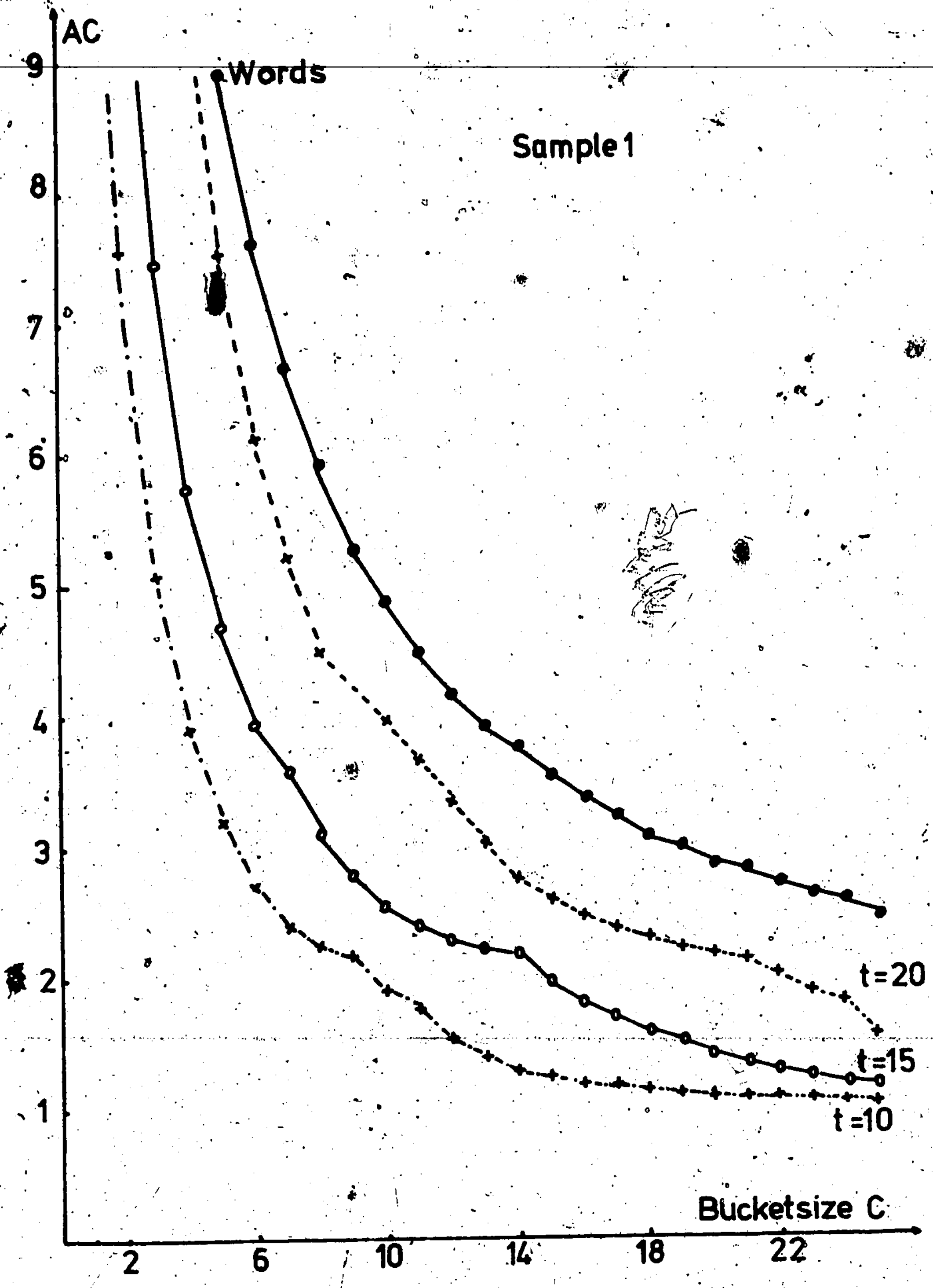


Figure 17

Average Number of Diskaccesses for Textfragment Dictionaries

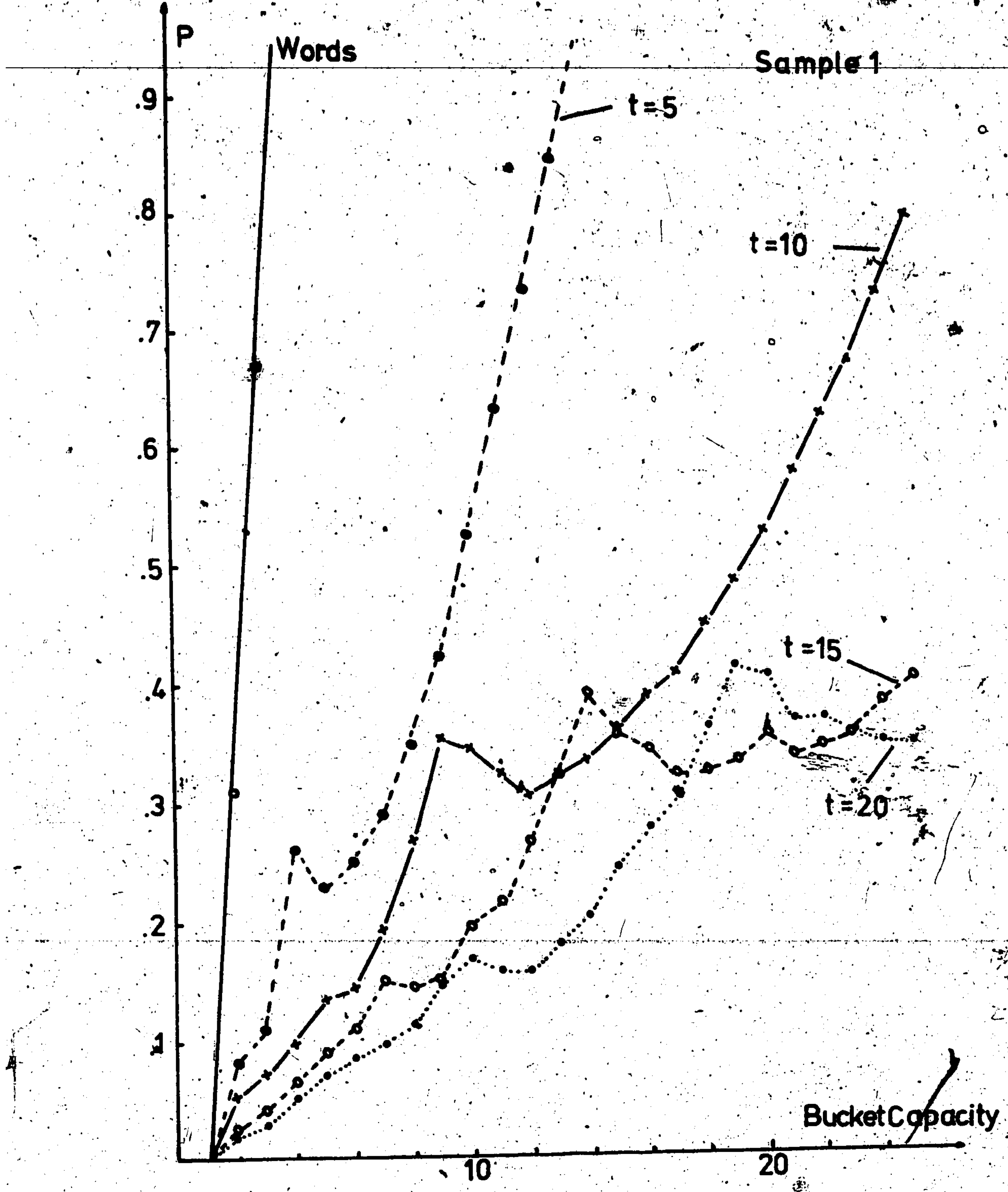


Figure 18

Memory Utilization Factor for Wordfragment Dictionaries

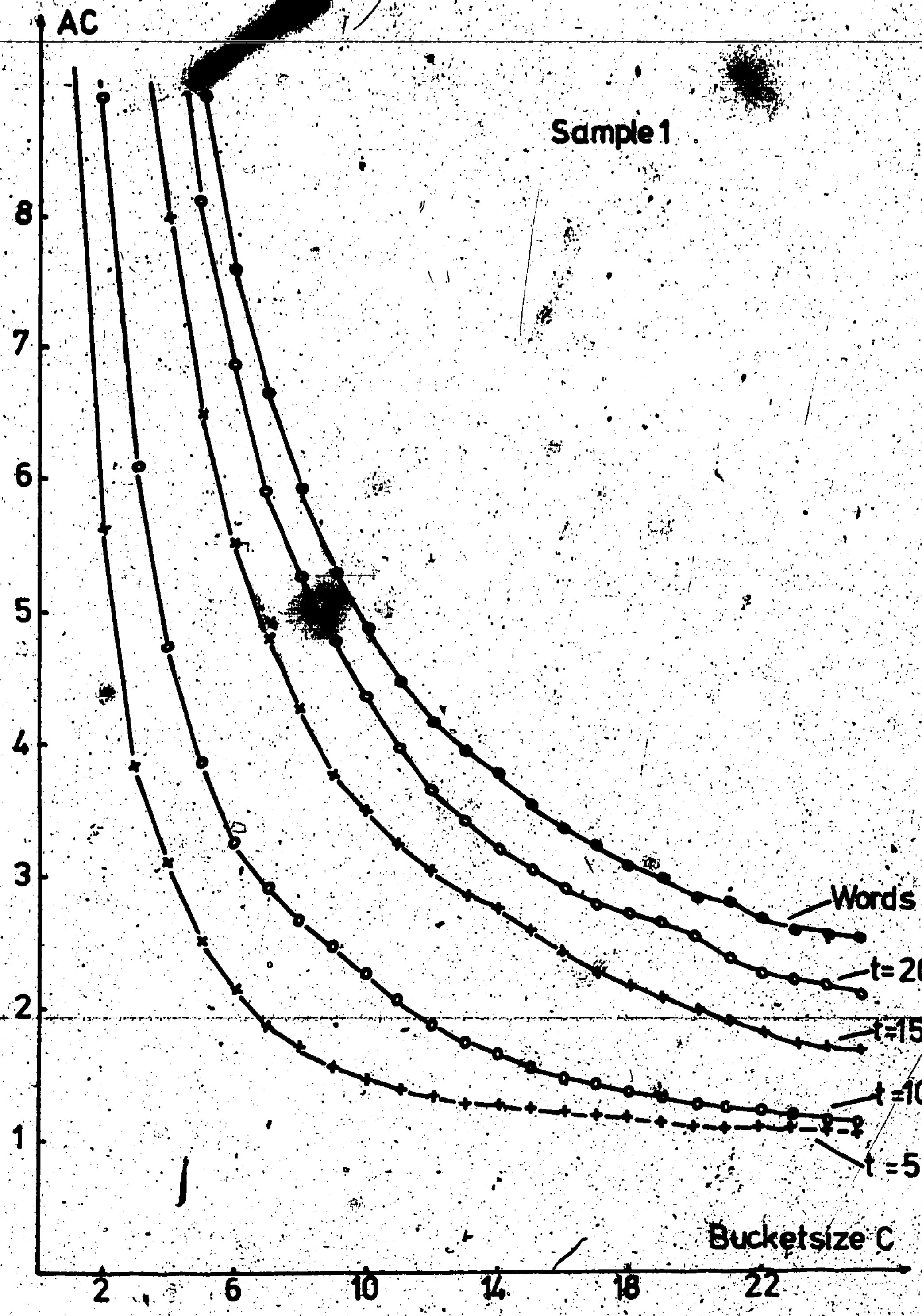


Figure 19

Average Number of Diskaccesses for Wordfragment Dictionaries

CHAPTER V

EFFECT OF FRAGMENT USAGE ON SYSTEM GENERATION, COMPRESSED DATABASE AND INVERTED INDEX

5.1 Decoding and Structure of Compressed Records

In a word oriented retrospective search system the compressed database records are the concatenation of variable length codes. These commafree codes allow a reconstruction of the original record, since each complete codeword implies a separating blank in the decoded record. This is the advantage of using the natural and well defined unit, the word, as the basic language element for compression coding. The disadvantage is the variable length code, forced by the Zipf distribution, which complicates the decoding process.

The approach taken by Thiel and Heaps [87] divides the codeword into two parts. The first part is an index into a table of pointers, the second a "string index" into a termstring. The table entry contains a pointer to the appropriate termstring from which the right term is retrieved by locating it with the string index. The described process is time consuming and in general requires a disk access for each term, unless the most frequent words are stored in core. Since decoding is a frequent operation special emphasis must be placed on efficient decoding.

The advantage of using fragments [illegible]ead of words is that fixed length codes result from the equifrequency property. Use of a fixed length code reduces the coding operation to a simple table lookup with the codeword as the index. If the dictionary is small enough to be core resident during decoding no disk accesses are necessary. In an on-line system this can result in considerable improvement in total system performance. The switch in language elements benefits decoding, but also affects the structure of the compressed records. Consideration will now be given to use of either word or text fragments.

The simple case of text fragments will be discussed first. With text fragments the compressed records are the concatenation of fixed length codes, and the original document record is the concatenation of the variable length strings corresponding to the codewords. Allowing embedded blanks in the text fragments eliminates the need for storage of extra markers to delimit the individual words. Blanks are either part of a text fragment, or a single character blank is stored to indicate the gap between two successive words. The use of text fragments combines some of the advantages of words and fragments. Records are still the concatenation of commafree codes, and an advantage over word coding systems, is that the codes are of fixed length. Furthermore there is no storage overhead in the compressed record.

However, the advantage of no overhead is lost when word fragments are used as language elements. The boundary markers between words must then be indicated, since word fragments do not contain any blanks. One word can consist of a single codeword or it can be the concatenation of two or more fragments. Three different ways of inserting delimiting blanks are shown in figure 20. Before the three schemes are described [illegible] detail it may be remarked that equation 3.3(2) speci[illegible] the length of the compressed database when text fragments are used. This equation is not valid for word fragments since the extra delimiters introduce storage overhead.

It will be supposed that the uncompressed database consists of a total of TC non-blank characters arranged to form a total of T word tokens. Suppose the word fragment dictionary contains ND fragments with an average fragment length of AF. The values of AF and ND are functions of the threshold. The total number of word fragments in the compressed database is therefore

$$NF = TC/AF \; . \qquad 5.2(1)$$

Set the length of the codewords for the fragments to

$$d = \mathrm{ceil}(\mathrm{ld}(ND)) \; .$$

A) Word Boundary Markers

The first scheme shown in figure 20 reserves a special codeword as a word boundary marker which is stored in the appropriate places of the compressed record. During decoding this special codeword causes the insertion of a blank in the decoded string. This approach is wasteful and prevents the efficient use of fixed length codes, since the blank is the most common character with a probability of approximately 0.2. An efficient code would assign a shorter codeword to the marker than to all other fragments. Since a marker of d bits is inserted after each word the size of the compressed database is given by

$$LCW = NF * d + T*d .$$

Using equations 5.2(1) and 3.2(1) the following relation can be derived

$$LCW = TC *d (1/AF + 1/AW) . \qquad 5.1(2)$$

The overhead, defined as the ratio of the number of bits TC*d/AW occupied by word delimiters to the number of bits TC*d/AF occupied by the fragments without delimiters, is given by

$$OV = AF/AW .$$

It must be remembered that AF is less than AW in the case of word fragments and that AF is a decreasing function of t. The larger the average fragment length the larger the overhead. For sample one with its four thresholds of 5,10,15 and 20 the overhead would be 55,48,45,43 percent respectively.

B) <u>Directory of Pointers</u>

Each record can be considered as a sequence of variable length fields, with each field representing one word. A common approach for the treatment of variable length fields is to store at the beginning of each record a directory that contains fixed length pointers to the start of each field. A directory of pointers to the codeword which represents the last fragment of a word is shown in figure 20, and it can be seen that this preserves the advantage of fixed length codes. The number of bits needed for the storage of one pointer must be sufficient to treat the case in which the longest record in the database, say of length R, is a concatenation of single characters. This yields for the length of one pointer

$$k = \mathrm{ceil}(\mathrm{ld}\ (R)) ,$$

and for the length of the compressed database

$$LCW = NF * d + T * k .$$

This can be rewritten, using equations 5.2(1) and 3.2(1), as

$$LCW = TC * d * (1/AF + (k/d)*(1/AW)) . \qquad 5.1(3)$$

The overhead is given by

$$OV = (k/d) * (AF/AW) .$$

For a system in which $k<d$ the overhead is smaller than with boundary markers. A dictionary with more than 512 elements and a document database which does not contain abstracts should satisfy $k<d$. For comparison, the values for the overhead of sample one using this approach, were calculated as 40,36,38,36 percent, and are considerably lower than in scheme A.

C) <u>Tagged Codewords</u>

In the last approach investigated, all codewords are tagged to identify the position of the fragment in the word. As seen in figure 20 each codeword can be tagged with an extra bit, which is set to one only if the codeword represents the last fragment of a word. It is convenient to use the first bit for this purpose since the remaining d-1 bits can be used as an index into the dictionary. This scheme preserves the fixed code length advantage, but adds the inconvenience of bit manipulation. The calculation of the length of the compressed database yields

$$LCW = (TC/AF)*(d+1), \qquad 5.1(4)$$

from which the overhead may be calculated as

$$OV = 1/d .$$

This result represents the smallest overhead of the three different storage modes as shown by the illustrative values of 9,10,10,11 percent that result for sample one. It is interesting to note that the overhead in this scheme is independent of the average fragment length.

The third scheme is clearly the best and therefore it is worthwhile to express LCW and OV as a function of t by use of the empirical equations found in chapter four.

Substituting 4.5(1) and 4.5(2) into 5.1(4) yields

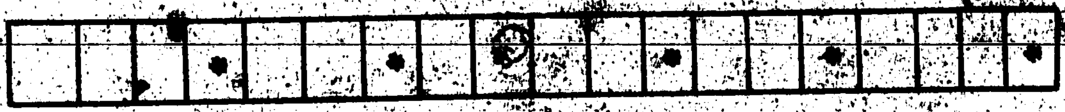
$$LCW = (TC/A0)*t**a(1 -ceil(ld\ N0 - n*ld(t)) , \qquad 5.1(5)$$

and for the overhead

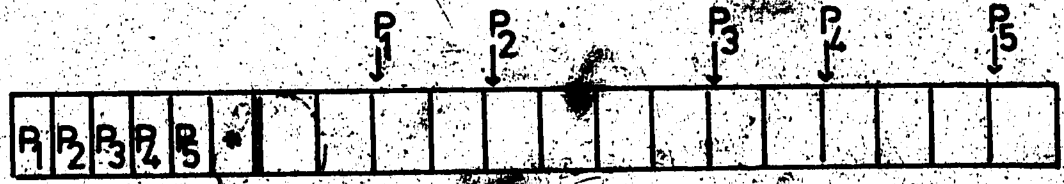
$$OV = 1/ceil(ld\ N0 - n*ld(t)) . \qquad 5.1(6)$$

Since LCW and OV are increasing functions of the threshold, it may be concluded, that a low threshold is the best choice to achieve good compression.

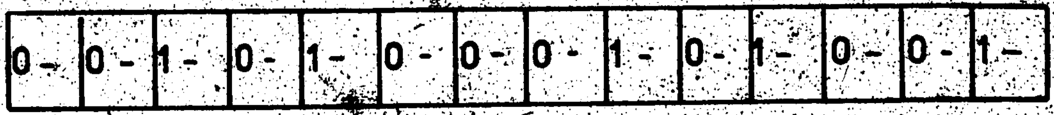
When comparing text and word fragments as above it is seen that text fragments are superior since they do not introduce any overhead. Practical results obtained with the two alternate schemes will be provided later.



Records with Markers



Records with Directory



Records with tagged Codewords

Figure 20

Storage Modes for Records coded with Wordfragments

5.2 Construction of the Inverted Index

One component of the retrieval system, the inverted index, is affected substantially by the change in language elements. The disparate Zipf distribution of words is replaced by the more uniform one of the fragments. This leads to an increase in the number of entries as derived in equation 3.3(5).

However, the real increase in the number of entries in comparison to a word oriented index is larger than specified by 3.3(5), since in the word system some of the elements eliminated are ones for which index entries should be made. All words on the stoplist are neglected when making index entries.

The disadvantage in the use of a word oriented inverted file governed by the Zipf distribution is most apparent when trying to organize the index on the disk. During system generation each term causes the allocation of a fixed size bucket, some of which fill quickly with entries. New buckets should be allocated to contain the spillover and should be adjacent to the filled one. This requirement necessitates constant reorganization of the index during system generation.

The uniform list length for a fragment oriented index represents a considerable advantage when organizing the index. The storage allocation for the index is simple since each dictionary fragment is assigned to a fixed size bucket of proper size. This bucket is supposed to hold all the record identification numbers of the documents containing the particular index fragment. If it is assumed that one fragment is distributed evenly throughout the database, an assumption made in the introduction, then all buckets will fill up at a constant rate. During coding (when the index entries are made) no bucket overflow is encountered and no reordering is necessary, provided the bucket size was suitably chosen at the beginning.

The storage for the fragment inverted index can be reduced by designating certain fragments for which no entries are to be made in the index. The decision as to which fragments should not be index fragments cannot be based on the frequencies, as it can with words, but must be made on the basis of suitability of the fragments for retrieval. Obvious candidates for elimination from the list of index fragments are those that bridge two words or contain words which would be in a word stoplist. Examples are "A_THE_BE","AND_THE_","OF_THE_I" and others. Similar ones can be found among the word fragments. A "fragment stoplist" can be prepared manually since the number of dictionary fragments is small. The dictionary elements not on this stoplist are the index fragments.

Neglecting the fragments stored on the stoplist allows the expression of the size of the inverted index as a function of the threshold t. This can be achieved by using the empirical relations found in chapter four. There are

ND(t) dictionary fragments with an average frequency of F'(t) in the dictionary. Let R be defined by equation 3.2(6). When the index is packed its minimum storage requirement is given by

$$IW(t) = ND(t) * F'(t) * R . \qquad 5.2(1)$$

Substituting the empirical relations from chapter four into 5.2(1) yields IW as a function of t as

$$IW(t) = N0*F0*R*t**(f-n) . \qquad 5.2(2)$$

Let each dictionary fragment occur with a frequency of f(j,t) with j = 1,...ND(t) and

$$\sum_{j=1}^{ND} f(j,t) = ND(t)*F'(t) .$$

Assume that fixed size buckets of size C*R bits are allocated for the index and no packing occurs. The storage requirements for the unpacked index are

$$IU(t) = C*R \sum_{j=1}^{ND(t)} ceil(f(j,t)/C) ,$$

which yields for the memory utilization factor P

$$P(t,C) = C* \sum_{j=1}^{ND(t)} ceil(f(j,t)/C)/(ND(t)*F'(t)) - 1 .$$

Derivation of these equations has not taken into account a possible duplication of index fragments within one record. The same index fragment may occur more than once in a record as part of different words. One index entry for this document is sufficient and elimination of these duplicates reduces storage requirements. The equations above can be adjusted by the introduction of a reduction factor based on the statistics of the database.

This reduction factor can be estimated if it is assumed that the distribution of fragments is multinomial. This assumption is valid if the probability of occurrence of a particular fragment is 1/ND(t) and a record is regarded as obtained by independent draws of a fragment from the full dictionary. The requirement of independent draws is quite strong and does not agree entirely with reality. It is quite common that some title words also occur in the subject heading. This leads to more duplication of fragments per record than predicted by the multinomial distribution and the predicted reduction factor must therefore be considered as a lower bound.

Let each fragment j occur with a multiplicity of f(j) in each record so that each record consists of the average of

$$NFR = \sum_{1}^{ND} f(j)$$

fragments. Obviously most of the f(j)'s are zero. With the assumption that the fragments follow a multinomial distribution, the probability for a particular configuration is given by :

$$p(f(1),\ldots,f(ND)) = \frac{NFR!}{f(1)!\ldots\ldots f(ND)!} * (1/ND) ** NFR \quad . \qquad 5.2(3)$$

Define the reduction coefficients c(j) of fragment j as

$$c(j) = \begin{cases} 0 & \text{if } f(j) = 0 \\ f(j)-1 & \text{if } f(j) > 0 \ . \end{cases}$$

The reduction factor is the ratio of index space after elimination of duplicates to the index space before elimination. The expected reduction factor RF is therefore

$$RF = (1/NFR)\sum (p(f(1),\ldots,f(ND))*(NFR - \sum_{1}^{ND} \delta(j)), \qquad 5.2(4)$$

where the first summation is over all possible permutations of all partitions of NFR. The expression can be reduced to a single sum over all partitions if the number of combinations for one partition is calculated as below.

Let $\sum q(k)*k = NFR$ represent a partition P of NFR with m nonzero q(k)'s. Each partition generates

$$d(m,P) = \frac{NFR!}{m!} * \frac{1}{\prod_{\substack{k=1 \\ q(k)>0}}^{NFR} q(k)!},$$

permutations. Inserting this into 5.2(4) yields:

$$RF = \frac{1}{NFR} * \frac{NFR!}{\prod_{q(k)>0}^{NFR} q(k)*k!} * \sum d(m,P)\,(NFR - \sum_{k=1}^{NFR} q(k)\,(k-1)). \qquad 5.2(5)$$

The summation now extends only over all partitions of NFR. This still represents a considerable amount of calculation, because of the inherent combinatorial problem of determing all partitions of NFR. To obtain an estimate of the magnitude of RF the above expression was evaluated for different values of NFR and ND, and the result is displayed in figure 21. To generate the partitions of NFR, Algorithm 95 from the Communications of the ACM was used [85]. Not surprising is the fact that the greatest reduction occurs for large records and a small dictionary. In a later section, when it is possible to compare theoretical with practical results it will be seen how realistic is the assumption of independence.

Construction of the inverted index occurs at the same time as the database is compressed. The algorithm which locates those fragments, which by concatenation make up the document, can also enter the document references in the index. For single letters or fragments on the stoplist no entries are made. The proper bucket is located by pointers associated with each fragment, and counters may be used to keep track of how the buckets fill up. In the event of bucket overflow, the best method for containing the spillover is to chain an overflow bucket to the filled one. The occurrence of overflow is a direct function of the algorithm used for compression. If the algorithm preserves the equifrequency property well, the probability of overflow is small. On the other hand if this property is not well preserved the probability of overflow is high and less favorable results are obtained. It is necessary to match the coding algorithm to the goals of the selection algorithm. The real measure, however, is provided by the results produced by the actual coding of the sample database.

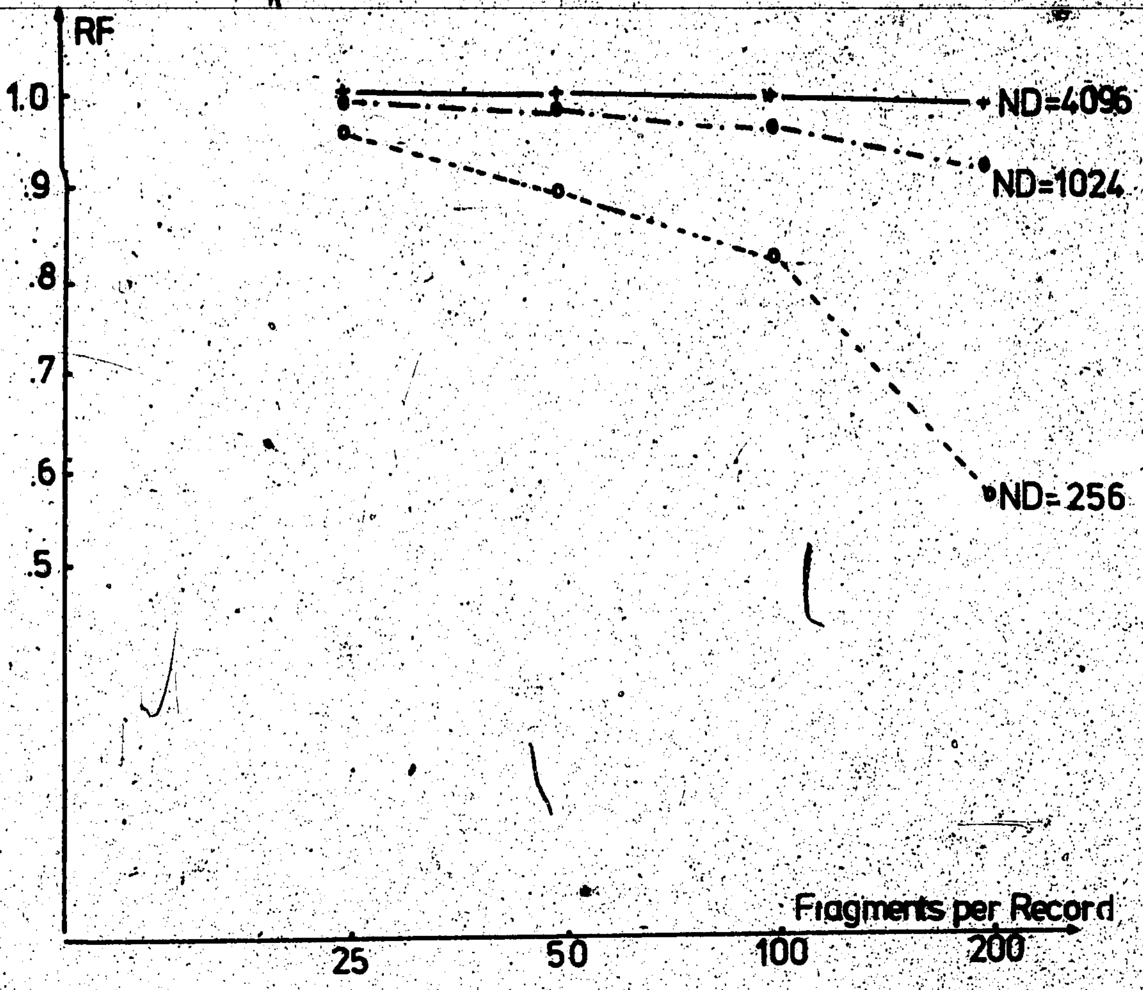


Figure 21
Predicted Reductionfactor for Inverted Index

5.3 Coding Algorithms

After the determination of fragments which form the dictionary, attention must focus on the coding method used to obtain the compressed records. An algorithm must be developed which selects from the dictionary the strings which, when concatenated make up the record. It is obvious that there exists more than one such algorithm to achieve the desired objective, and therefore criteria for comparison of the algorithms must be stated. The algorithm which yields the best performance as measured by these criteria is the one that should be chosen for the compression. It must be kept in mind that it is sufficient to choose the algorithm with the best performance within the constraints imposed by the physical resources of the computer system. This implies that the one selected for the job is not necessarily the one with the best overall performance.

Fortunately these criteria are easy to establish since the use of fragments has two main objectives. These are "good" compression and "efficient" use of disk space allocated for the index. If it is assumed that the selection of dictionary fragments produced a nearly optimal set then this set can be used as a basis for comparison. If, after coding of the database, the dictionary fragments have the same frequencies as predicted by the selection algorithm then the result is optimal. Any distortion in the frequencies of the dictionary fragments after coding will result in less optimal performance. The real problem is to match the coding algorithm to the selection algorithm so that they produce the same frequency distribution. This is a nontrivial matter since the selection algorithm was based on the advance knowledge of the whole database or its representative sample.

The coding algorithm does not know its total input in advance, it sees only one record at a time. It is therefore very difficult to make a decision on which of the many fragments contained in the record to choose since the effect on the final distribution is known only after the entire database has been coded. A backtracking and changing of records already coded is impossible.

The criteria for comparison can be divided into three groups. One is concerned with compression, one with the index, and the third with the algorithm itself. The main emphasis is on the first two groups since a coding algorithm which leads to a retrieval system with a poor performance is unacceptable, even if it is the best according to some criterion. Some inefficiency in the coding algorithm can be tolerated, since coding is a one shot operation whereas decoding is a more frequent operation.

From section 2.2 it is known that there are two measures of compression. The measure of special significance is the compression ratio and its inverse the compression ICR. The smaller ICR, the better the compression. ICR has a direct practical meaning; it represents the fraction of the original database storage needed by the compressed version. The other more theoretical measure is the efficiency E as defined by 2.3(3). This measure shows how well the compression code matches the frequencies of the coded terms. It yields an absolute comparison with the best possible compression based on the same language elements and the same frequency distribution. For the calculation of these two measures all fragments in the dictionary must be used, even the single letters. The main emphasis here is on ICR but some attention must also be devoted to E.

The two quantities ICR and E are not as important as those which involve the index, since the compressed database is normally stored on a cheap medium such as magnetic tape. The storage medium for the index is likely to be more expensive. The main focus is therefore on the inverted index measures since it affects the total system cost much more and a deviation from optimality is more serious.

Three index related quantities exist and can be used as a measure. Two of these quantities are of practical significance, the disk utilization factor P of equation 3.2(9) and the average number of disk accesses AC. These two quantities are based on the assumption that fixed length buckets are assigned to each index element and that each element has a separate bucket. Since P and AC are functions of the bucket size C graphs of P and AC versus C may be used to display the results. In the calculation of P and AC only index fragments should be used, which means the exclusion of single letters.

The third quantity besides P and AC is the efficiency E. It can function as a measure of equifrequency, the goal set for the index fragments. Again only index fragments must be used to calculate E and for the remainder of the chapter this index efficiency is denoted by E'.

The two previous groups of measures are important for the operation of the system but they do not measure the efficiency of the coding algorithm itself. The only quantity that measures the performance of the algorithm is the time required to code the database and its claims made on core memory during coding. The next sections will describe the investigation of three different algorithms. Experimental results will show that the coding time is the criterion that discriminates the best between the three algorithms.

5.3.1 MS-Algorithm

The algorithm which yields the minimum storage form for the compressed version of the database is of special interest because of the emphasis put on the length of the compressed database. The database consists of individual records and, since there is a logical break between records, each record must be coded separately. Coding the database is thus reduced to coding of the individual records. Each record must be represented by concatenations of fixed length codes that replace the selected variable length dictionary fragments occuring in the record. The minimum storage form is the one which requires the smallest number of codewords. The minimum storage form of the whole database is the one where all records are in minimum storage form. Thus the problem is reduced to finding the minimum number of codewords necessary to represent a single record without information loss.

This problem has been investigated by Wagner [90] in connection with compiler construction and was formulated as an integer programming problem. Wagner also provides a dynamic programming algorithm to solve the problem [91]. The described algorithm has the remarkable property that the coding time is a linear function of the length of the record to be coded.

However, the use of this algorithm was not investigated since an alternate solution to the minimum storage problem was found before the appearance of Wagner's article. It is possible to reduce the above problem to one well known in optimization theory, which is that of finding the shortest path between two nodes of a network. For this problem a set of nodes and links (edges) between these nodes is given and two nodes are designated as start and end node. Each link has an associated cost which is normally the length of the path. Desired is the sequence of nodes and edges connecting start and end node with minimum cost.

It can be shown that with suitable interpretation the minimum storage form and the shortest path problem are equivalent.

Assume that the r letters in a record (word) are numbered consecutively from 1 to r and that each letter represents a node in a graph. Assign node r+1 to the blank following the record. A k-letter fragment starting at letter i can be considered a path from node k to node k+i. Assign each link a cost of one regardless of the value of k. All these links are directed with the edge always pointing from the lower numbered node to one with a higher number. To locate all the edges in the r+1 node graph all fragments contained completely in the record must be found. Each link represents one codeword, and the associated shortest path problem can now be solved. The minimum number of links is the same as the minimum number of codewords for the coded record. The sequence of edges in the shortest path yields the fragments for the minimum storage form.

An example of the MS-problem in a form suitable for a shortest path solution is shown in figure 22. It can be seen that the inclusion of the single letters in the dictionary guarantees at least one path between start and end node, so that the graph is not separated and a solution always exists.

The advantage of reducing the MS-problem to a shortest path problem lies in the fact that it is a well understood and well studied problem for which alternate solutions exist. A survey of these methods is given by Pollack, et al.

[62]. A heuristic method to find the shortest path was given by Nicholson [55]. From the data provided it can be seen that this method is superior to any linear or dynamic programming algorithm. For this reason this method was selected for the testing. The algorithm used is the one published by Boothroyd [6] based on Nicholson's paper.

The representation of the links between the nodes of an r-letter record is normally done by an adjacency matrix of size (r+1) x (r+1). If only fragments of maximum length p are used, and the fact that all links are forward is taken into account, then an (r+1) x (p+1) matrix is sufficient.

The problem of locating all fragments contained in the record can be solved easily. To find the single letters is trivial. Longer fragments can be located by using bigrams to hash into the root nodes of a tree-like dictionary and then proceeding with a tree search. It is obvious that a large amount of time will be spent in searching the dictionary.

Unfortunately Nicholson did not derive any results to relate execution time to the number of nodes in the network. Hence execution time dependency on the length of the input record is not known. Results of the application of this shortest path solution method to the minimum storage problem are presented in section 5.4.

Sample Word

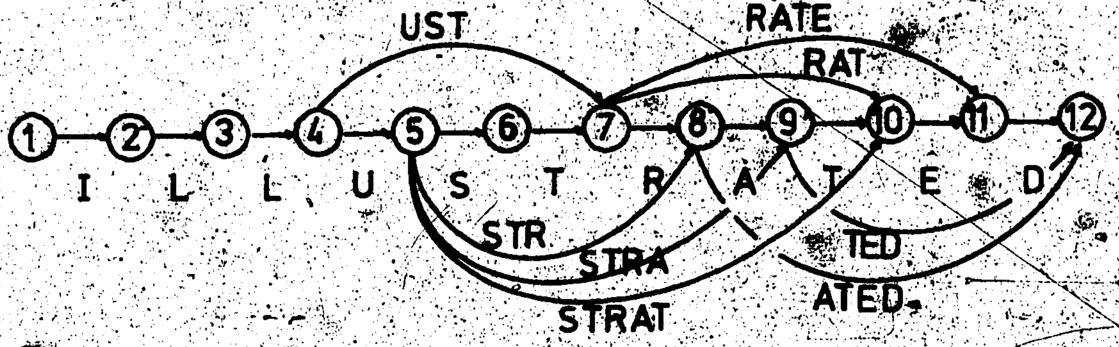
Letters: I L L U S T R A T E D _
Nodes: 1 2 3 4 5 6 7 8 9 10 11 12

Fragments of length ≥ 2 contained in "ILLUSTRATED":

UST, STR, STRA, STRAT, RAT, RATE, ATED, TED

[Wordfragment Dictionary t = 10]

ASSOCIATED GRAPH



Possible Solutions:

I| L| L| UST| RATE| D
I| L| L| U| STRA| TED
I| L| L| U| STR| ATED

Figure 22
Graph Representation of Minimum Storage Problem

5.3.2 <u>LFF-Algorithm</u>

The MS-algorithm presents the optimal solution for compression purposes, but since it is not known how well it preserves the equifrequency, an alternative must be studied. The known drawbacks of the MS-algorithm are its complexity and the amount of time needed for coding. The alternate scheme investigated can be called a <u>l</u>ongest <u>f</u>ragment <u>f</u>irst algorithm.

In this approach all fragments contained in the record must be located and stored similarly to the MS-algorithm. After all subfragments of the record are available the algorithm recursively removes the longest fragment. It deletes the longest dictionary fragment from the record and eliminates all those subfragments which overlap into the removed string. This process is repeated until the record is empty. At this point all fragments that make up the compressed record have been found.

Obviously this algorithm yields a different set of fragments than that produced by the MS-algorithm. It may have the same number of codewords, but likely it will have more. This will undoubtedly affect compression and it is necessary to study how serious this deterioration is. A possible alternative to this algorithm is investigated next.

5.3.3 LM-Algorithm

The two algorithms above have a common disadvantage. They must both locate all the subfragments of the record to be coded. This requires long searching for the subfragments and large auxiliary tables. Since both algorithms eliminate all fragments overlapping into an already selected string, the time to search for these fragments was wasted. To avoid this excessive searching a modification of the LFF-algorithm was investigated in the hope of substantially reducing coding time. This scheme can be called a longest match algorithm.

The algorithm operates in a simple recursive fashion. Starting with the first character of a record all fragments are found which match the first and subsequent letters of the record. The maximum length of a matching fragment is restricted by the length p of the longest fragment in the dictionary. Suppose the longest matching fragment has a length of $k \le p$. This fragment is used immediately to compress the record and the process is now repeated starting at position k+1 and continued until the end of the record is reached.

There is no need to look for all those fragments which might start at letters 2,3..k of the record and this should result in a considerable saving of coding time. Furthermore not all the fragments contained in the entire record must be found before a decision is made about which fragments to use. This scheme can operate with an auxiliary vector of length p and therefore has reduced core requirements in comparison to the other two algorithms. How well this algorithm compares with the two previous ones with regard to the specified criteria will be seen in the next section .

5.4 Results from Coding Experiments

The three algorithms described were applied to the original database of sample one and various statistics were gathered. Only sample one was used for the tests since these experiments are quite time consuming and expensive. The coding was performed with the various dictionaries generated by the algorithm of chapter four. In all these tests the implementation collected the same statistics during coding so that the overhead was the same for all experiments. This allows good comparisons between the different algorithms but does not indicate how much of it is overhead. The experimental results can be divided into two groups based on the dictionary used, whether they contained word or text fragments.

5.4.1 Text Fragment Coding of Records

For these test three different text fragment dictionaries were used corresponding to thresholds of 10,15 and 20. The dictionary associated with a threshold of five could not be tested because of the limited core memory available on the system used. However, the results of the tests with the three dictionaries are fairly indicative, so that the results of the one set can be dispensed with.

The input data for the three algorithms were all the database records, and each record was treated separately. The results of the coding experiments can be found in Table 11 and are represented by the criteria as described in section 5.3. For contrast, the nearly optimal values for the dictionary are also listed. A graphical representation can be found in figures 23,24,25, which for comparison also displays the "predicted" performance parameters of the dictionary. I(t) is the number of entries in the index, T(t) the total number of codewords in the database, L(t) the length of the coded database in bits and C(t) the time to code the sample measured in seconds.

When inspecting the results one thing is very noticeable. The figures produced by all three coding algorithms are substantially different from those of the dictionary and show a marked deterioration. After coding the average length of the fragments is lower by almost one character per fragment and the entropy is lower by almost two bits per fragment. The compression ICR is the most affected quantity, since both HF and AF influence the value of ICR. ~~The~~ performance of the three algorithms is about the same with regard to compression. However, the size of the compressed database is about twice the size as predicted by the nearly optimal dictionary. In general the MS and LFF algorithms are almost indistinguishable in the results they produce, while the LM-algorithm is generally worse than the other two. In the graphs some results were not plotted for all algorithms since inclusion of all results would obliterate the graph.

The quantities connected with the inverted index show a rather interesting behaviour. The efficiency E', as a measure of equifrequency, deteriorated by only about three percent. However, when looking at the number of index entries it is obvious that the number of entries is considerably lower than predicted. It indicates that the number of single characters in the database is much larger than one would expect from the dictionary statistics. Obviously none of the three coding algorithms leads to the increase in the number of index entries as predicted by equation 3.3(5). This might lead to lower recall of the retrieval system, a question to be studied later.

The two other characteristics of the inverted index, the storage utilization factor P and the average number of disk accesses are shown in figures 26 and 27 as a function of the bucket size C. For comparison the graphs for word coding and the dictionary are also shown.

With regard to the values of P and AC all three algorithms yield only slightly different results. Again it can be seen that the performance is well below the one predicted, but is far better than for words.

The last quantity left to discuss is the coding time C(t) as a measure of the algorithm itself. This is where the most significant differences occur. The MS-algorithm with its searching and backtracking feature consumes about twice as much time as the more straightforward LFF-algorithm. The LM-algorithm in turn consumes only half as much as the LFF-algorithm because of the elimination of the unnecessary searches. If computer time is a major factor, then considerable savings in coding time can result by use of the LM or LFF algorithm. The decrease in compression is not very significant between the three algorithms. The time required to code the sample database appears quite large, but it must be remembered that the coding operated with the considerable overhead of gathering statistics. All programs were written in FORTRAN and it is felt that the same algorithms programmed in a low level language would yield a much smaller coding time.

To illustrate the three different algorithms some compressed records generated by the three algorithms are shown in Table 12. The dictionary used is the one listed in Appendix I for text fragments with a threshold of ten.

	t=10	t=15	t=20	
AF	2.03	3.37	3.12	
HF	9.51	9.18	8.79	
E	.963	.971	.965	
F	27	22	30	
E'	.994	.993	.994	DICTIONARY
I	7946	12694	13247	
L	135320	151350	163500	
ICR	.331	.371	.400	
AF	2.03	1.94	1.84	
HF	7.00	7.02	6.82	
E	.710	.742	.749	
F	27	36	50	
E'	.963	.966	.969	MS
I	7946	8899	8836	ALGORITHM
T	25205	26377	27717	
C	520	574	610	
L	252050	263770	277170	
ICR	.617	.646	.679	
AF	1.99	1.91	1.82	
HF	6.88	6.92	6.73	
E	.698	.732	.739	
F	28	38	51	
E'	.966	.968	.967	LFF
I	7512	8485	8434	ALGORITHM
T	25628	26652	28060	
C	237	264	290	
L	256280	266520	280600	
ICR	.628	.65[illegible]	.687	
AF	1.92	1.84	1.77	
HF	6.89	6.93	6.76	
E	.699	.732	.742	
F	29	40	52	
E'	.959	.961	.967	LM
I	8059	9088	8974	ALGORITHM
T	26565	27691	28807	
C	156	174	197	
L	265650	276910	288807	
ICR	.650	.678	.707	

TABLE 11

SUMMARY OF RESULTS FROM CODING WITH TEXT FRAGMENTS

```
SOCIAL|_|SCIEN|CES|_STUD|Y_AND_T|EAC|HIN|G|.|

G|A|R|D|NER,_|MAR|T|I|N|

THE|_|U|N|E|X|P|E|C|T|E|D|_H|A|NGIN|G,|_AND_O|THER_|
MATHE|MAT|ICAL|_|D|IVERS|ION|S|.|

R|I|G|G,|_H._|K.|

T|ALE|S|_FROM_|THE_S|K|IP|P|E|R|

AMERICAN|_LITERATUR|E|_|20|TH_CENTURY|.|
```

<u>**MS-Algorithm**</u>

```
SOCIAL|_|SCIEN|CES|_|STUD|Y_AND_T|E|ACHIN|G|.|

G|A|R|D|NER,_|M|A|RTI|N|

THE|_|U|N|E|X|P|E|C|T|E|D|_|H|A|NGIN|G,|_AND_O|THER_|
MATHE|M|A|TICAL|_|D|IVERS|ION|S|.|

R|I|G|G,|_H._|K.|

T|ALE|S|_FROM_|THE_S|K|I|PP|E|R|

AMERICAN|_LITERATUR|E_|20|TH_CENTURY|.|
```

<u>**LFF-Algorithm**</u>

```
SOCIAL|_|SCIEN|CES|_STUDY|_AND_|T|EAC|HIN|G|.|

G|A|R|D|NER,_|MAR|T|I|N|

THE|_|U|N|E|X|P|E|C|T|E|D|_|H|A|NGIN|G,|_AND_O|THER_|
MATHE|MAT|IC|AL|_|D|IVERS|ION|S|.|

R|I|G|G,|_H._|K.|

T|ALE|S|_FROM_|THE_S|K|IP|P|E|R|

AMERICAN|_LITERATUR|E|_|20||TH_CENTURY|.|
```

<u>**LM-Algorithm**</u>

TABLE 12

CODING OF RECORDS WITH TEXT FRAGMENTS

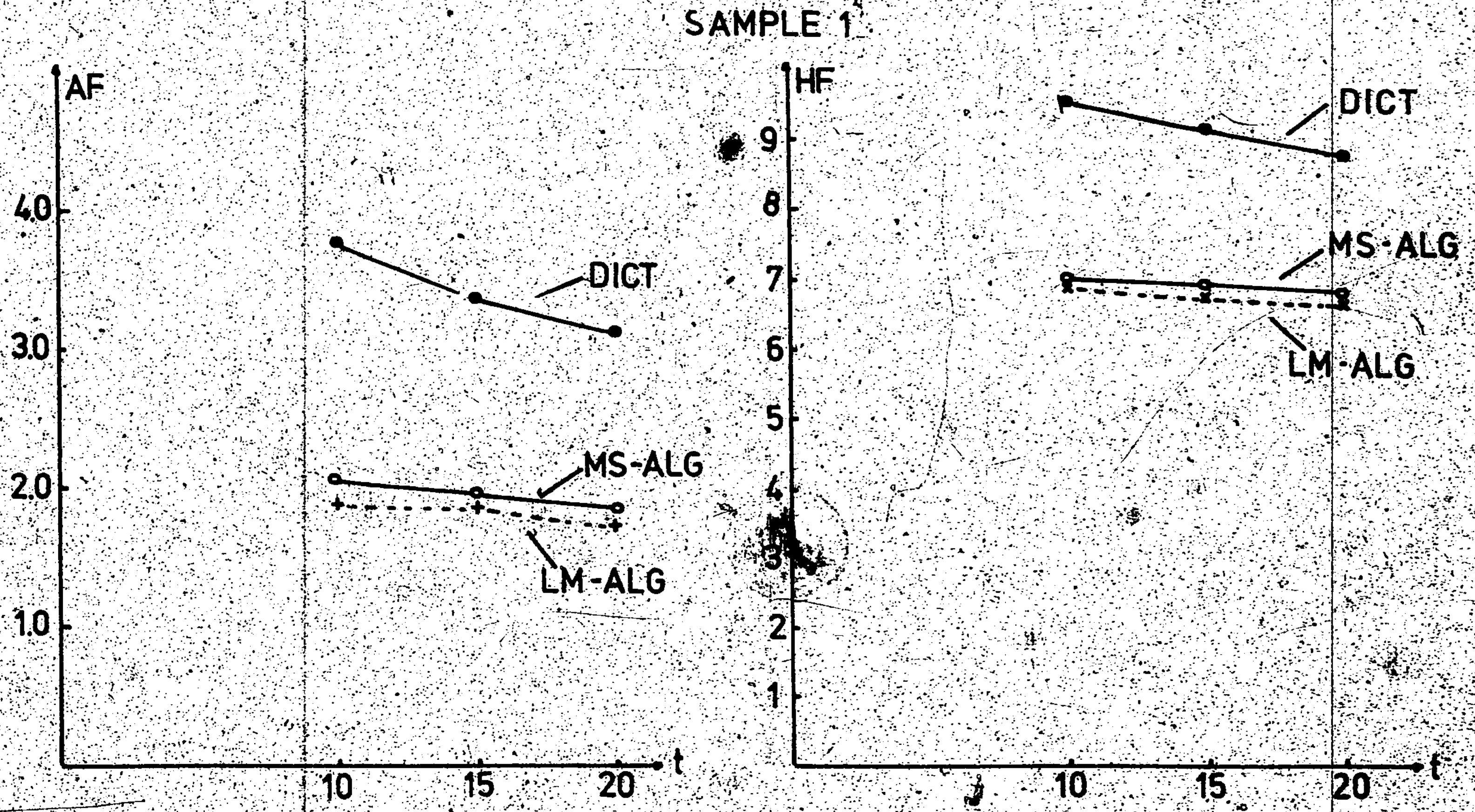


Figure 23

Average Fragmentlength and Entropy after Coding with Textfragments

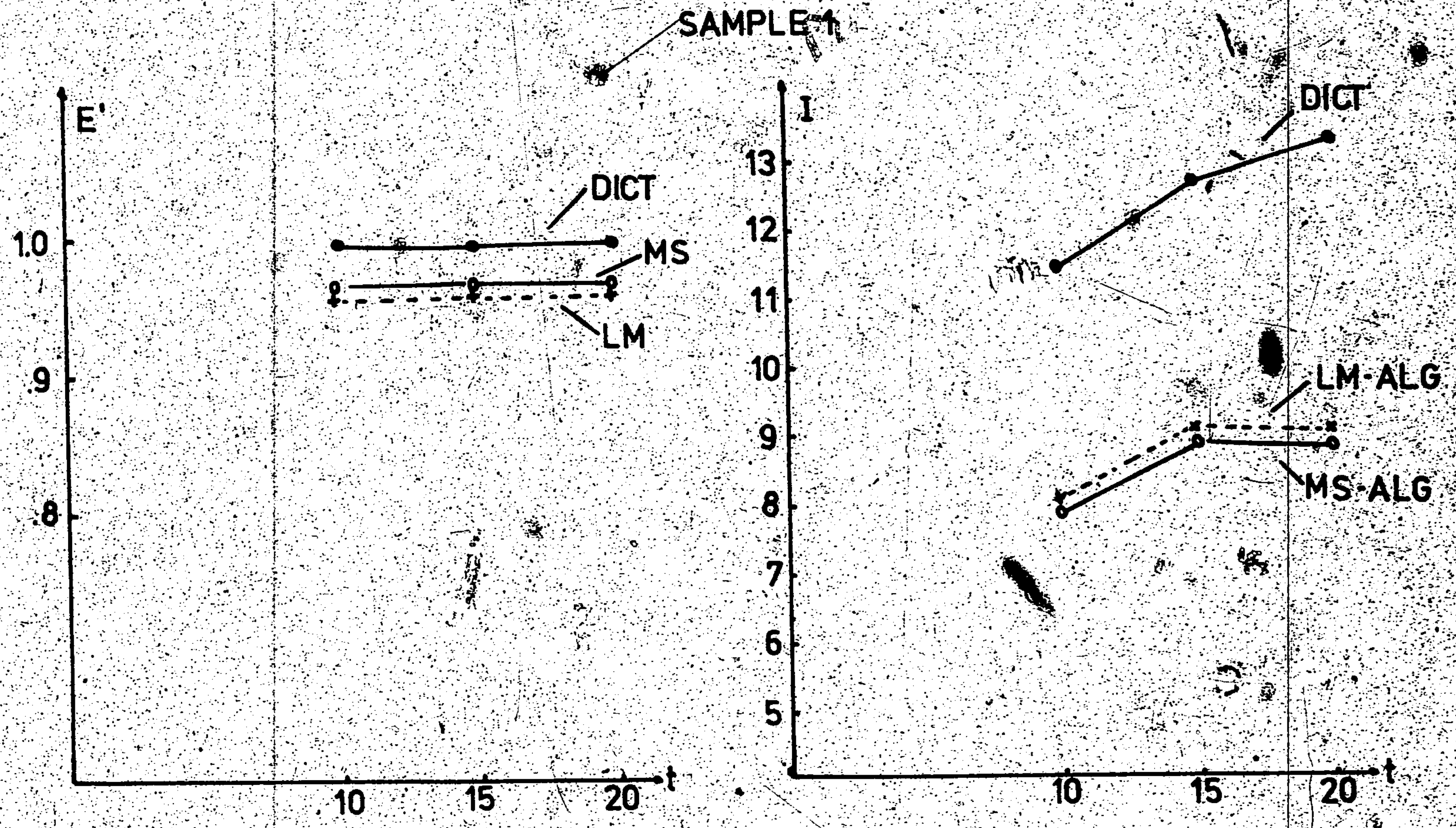


Figure 24

Efficiency and Number of Indexentries after Coding with Textfragments

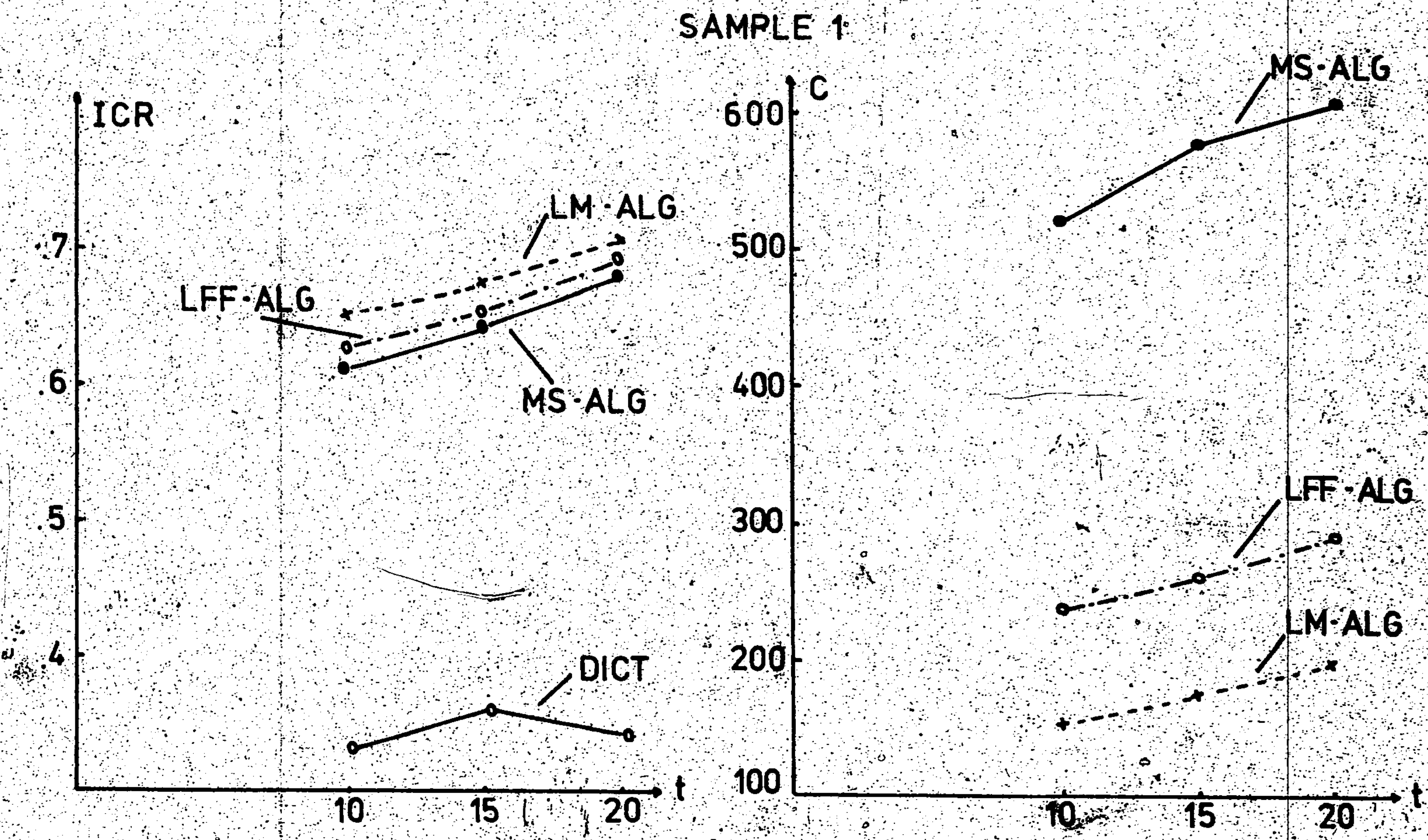


Figure 25

Compression and Coding Time after Coding with Textfragments

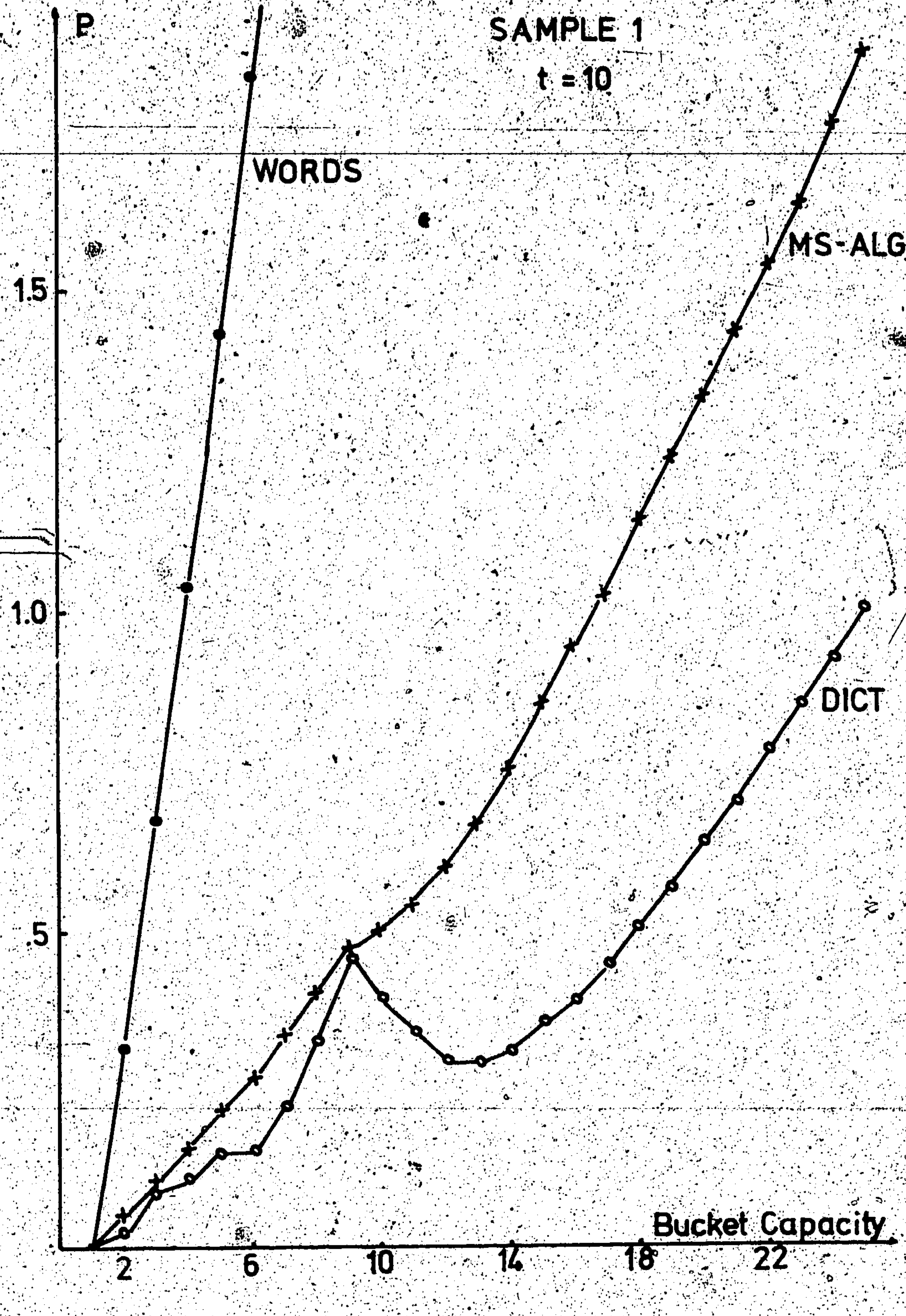


Figure 26

Memory Utilization Factor after Coding with Textfragments

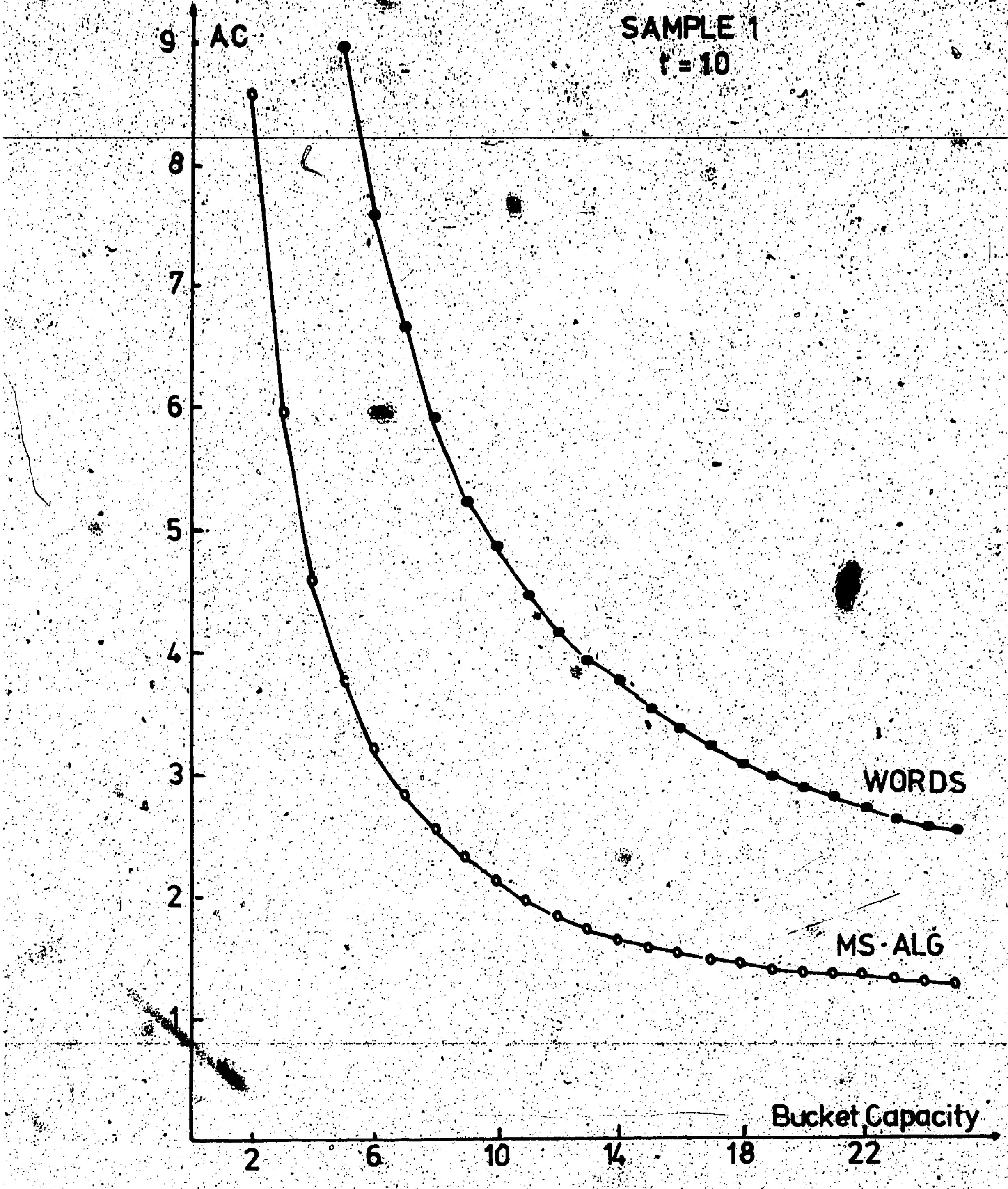


Figure 27

Average Number of Diskaccesses after Coding with Textfragments

5.4.2 Coding of Words with Word Fragments

Similar tests were run with the word fragment dictionaries corresponding to thresholds of 5,10,15 and 20. For these tests the input, i.e. the records to be coded, were the word frequency lists, so that computing time could be saved. The results of these experiments can be found in Table 13 in which the dictionary values are listed for contrast. A graphical representation is found in figures 28 through 30. The memory utilization factor P and the average number of disk accesses AC for the threshold ten are shown in figures 31 and 32.

The results concerning the compression ICR deserve some explanation. As mentioned in 5.1, word fragments require a special storage form for the compressed records which introduced storage overhead. The values of L(t) in the table do not contain any overhead. However, in the calculation of ICR the overhead generated by the use of tagged codewords was added to allow immediate comparisons with the compression obtained with text fragments as shown in Table 11.

In general the behaviour shown by the critical parameters is similar to that exhibited with text fragments. The MS and LFF algorithms are about the same and only the LM algorithm deviates considerably. The performance of the three coding algorithms is again well below the one predicted by the dictionary.

However, it may be noted that the coding times for the MS and LFF algorithms are approximately equal. This can be explained by the observation that both algorithms spend most of their time locating all the subfragments of the words. The searching and backtracking of the MS-algorithm plays a minor role since the words are considerably shorter than the full records. Again the LM-algorithm is the one that consumes the least amount of computer time. A sample listing from the coded word frequency list as processed by the three algorithms is given in Table 14. It shows that sometimes the LFF produces the same coding as the MS-algorithm, and in some cases the LM does also the same.

	t=5	t=10	t=15	t=20	
AF	3.10	2.75	2.59	2.47	
HF	9.10	8.69	8.38	8.19	
E	.871	.897	.908	.916	
F	1o	20	29	37	
E'	.982	.990	.985	.986	DICTIONARY
I	10195	11496	12424	12944	
T	14549	16548	17500	18319	
L	130491	164580	175000	164771	
ICR	.361	.449	.476	.453	
AF	2.10	1.88	1.78	1.72	
HF	7.37	7.14	6.93	6.84	
E	.706	.737	.750	.765	
F	16	29	42	53	
E'	.951	.964	.970	.956	MS
I	7548	8256	8353	8877	ALGORITHM
T	21581	23977	25409	26221	
C	85	95	109	117	
L	237391	239770	254090	235989	
ICR	.633	.651	.692	.648	
AF	2.08	1.87	1.77	1.72	
HF	7.24	7.04	6.85	6.72	
E	.694	.727	.742	.758	
F	16	29	42	54	
E'	.954	.964	.970	.956	LFF
I	7167	7936	8070	8582	ALGORITHM
T	21785	24219	25575	26342	
C	87	97	112	121	
L	239635	242190	155750	237078	
ICR	.640	.653	.690	.645	
AF	1.99	1.82	1.72	1.67	
HF	7.20	7.02	6.84	6.75	
E	.689	.725	.741	.755	
F	16	30	44	55	
E'	.940	.958	.965	.952	LM
I	7644	8291	8406	8927	ALGORITHM
T	22782	24925	26218	27173	
C	50	53	62	68	
L	250602	249250	262180	244557	
ICR	.670	.674	.706	.665	

TABLE 13

SUMMARY OF RESULTS FROM CODING WITH WORD FRAGMENTS

AGE|D	DRE|A|M	INTE|RIO|R
A|GES	E|LECTION	M|EM|ORA|N|DU|M
CAN|ADA	E|X|E|R|C|I|S|E|S	O|R|GA|NIZATION
CHANG|E	FO|R|MAL	P|E|RIO|D
CHA|N|GES	GAR|D|N|ER	ROBERT|SON
D|I|ARY	H|ARRI|S	STR|E|NGT|H

MS-Algorithm

AGE|D	DRE|A|M	INTER|I|O|R
AGE|S	E|LECTION	M|EM|ORA|N|DUM
CAN|ADA	E|X|E|R|C|I|S|E|S	O|R|GA|NIZATION
CHANG|E	FORM|A|L	PER|I|O|D
CHANG|E|S	GAR|D|N|E|R|	ROBERT|SON
DIA|R|Y	H|ARRI|S	STR|ENG|T|H

LFF-Algorithm

AGE|D	DRE|A|M	INTER|I|O|R
AGE|S	ELECT|I|O|N	M|EMO|R|A|N|DU|M
CAN|ADA	E|X|E|R|C|I|S|E|S	ORGANI|Z|A|T|I|O|N
CHANG|E	FORM|A|L	PER|I|O|D
CHANG|E|S	GAR|D|N|E|R	ROBERT|SON
DIA|R|Y	H|ARRI|S	STR|ENG|T|H

LM-Algorithm

TABLE 14

WORDS AFTER CODING WITH WORD FRAGMENTS

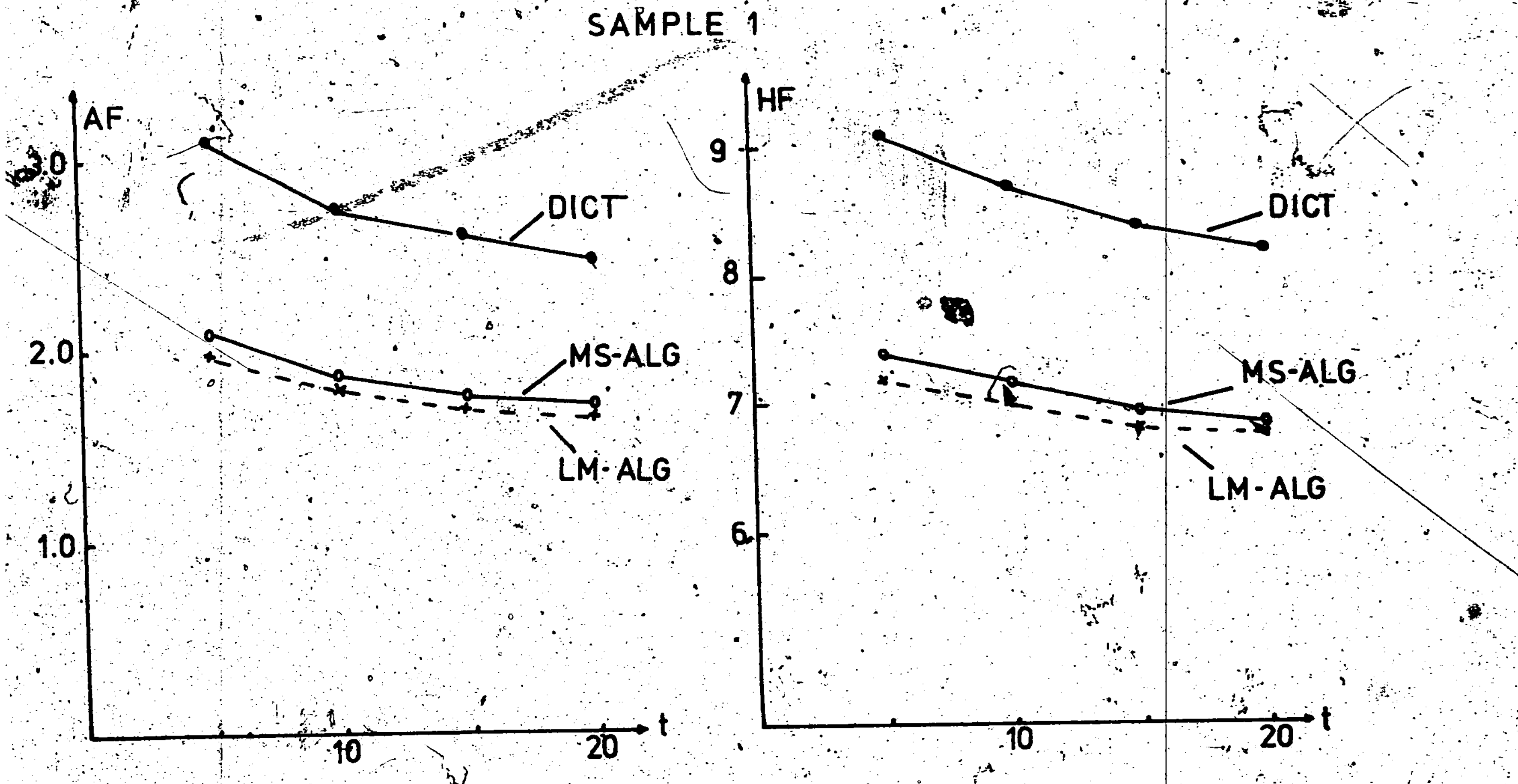


Figure 28

Average Fragmentle[illegible] and Entropy after Coding with Wordfragments

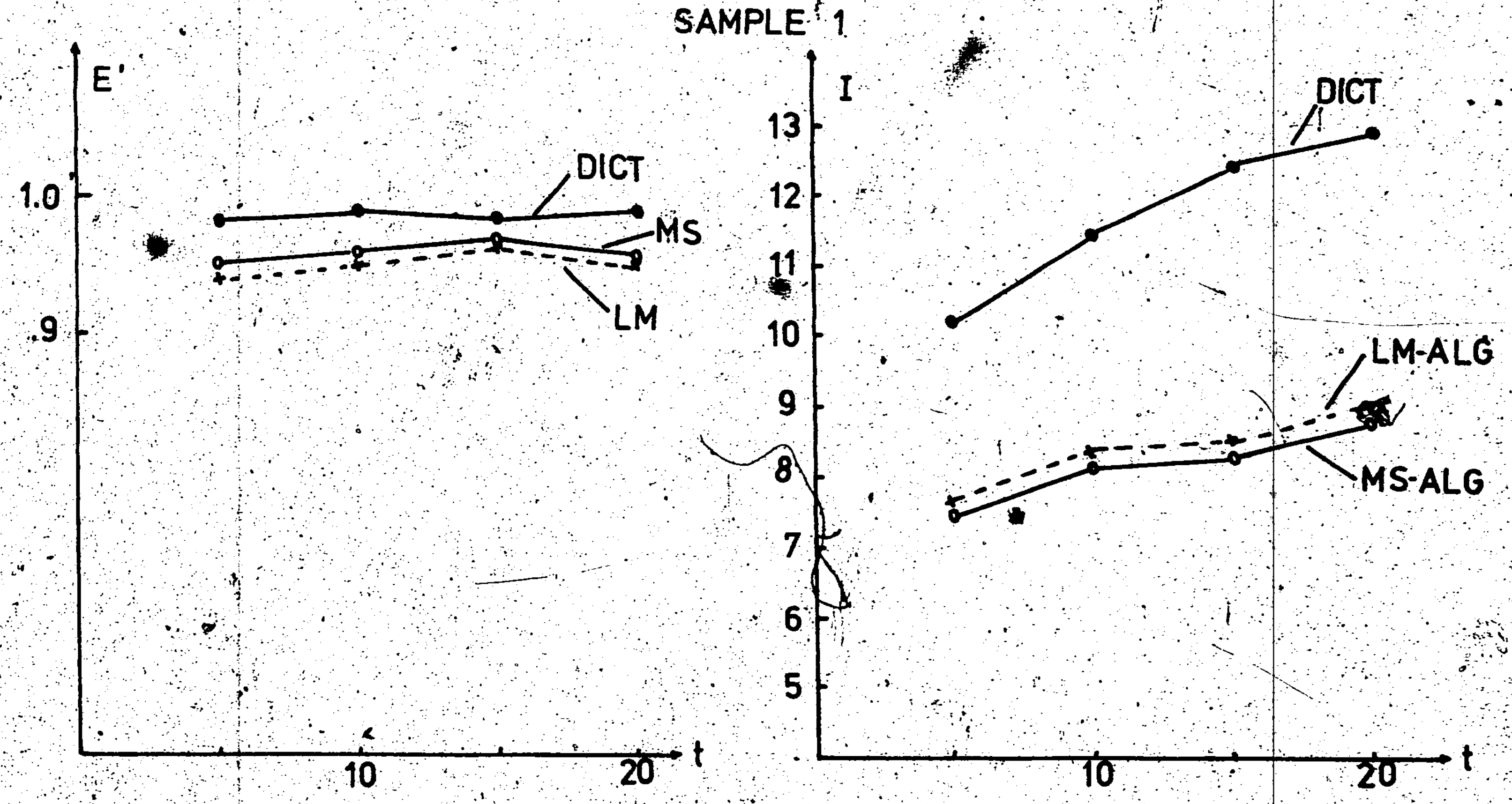


Figure 29

Efficiency and Number of Indexentries after Coding with Wordfragments

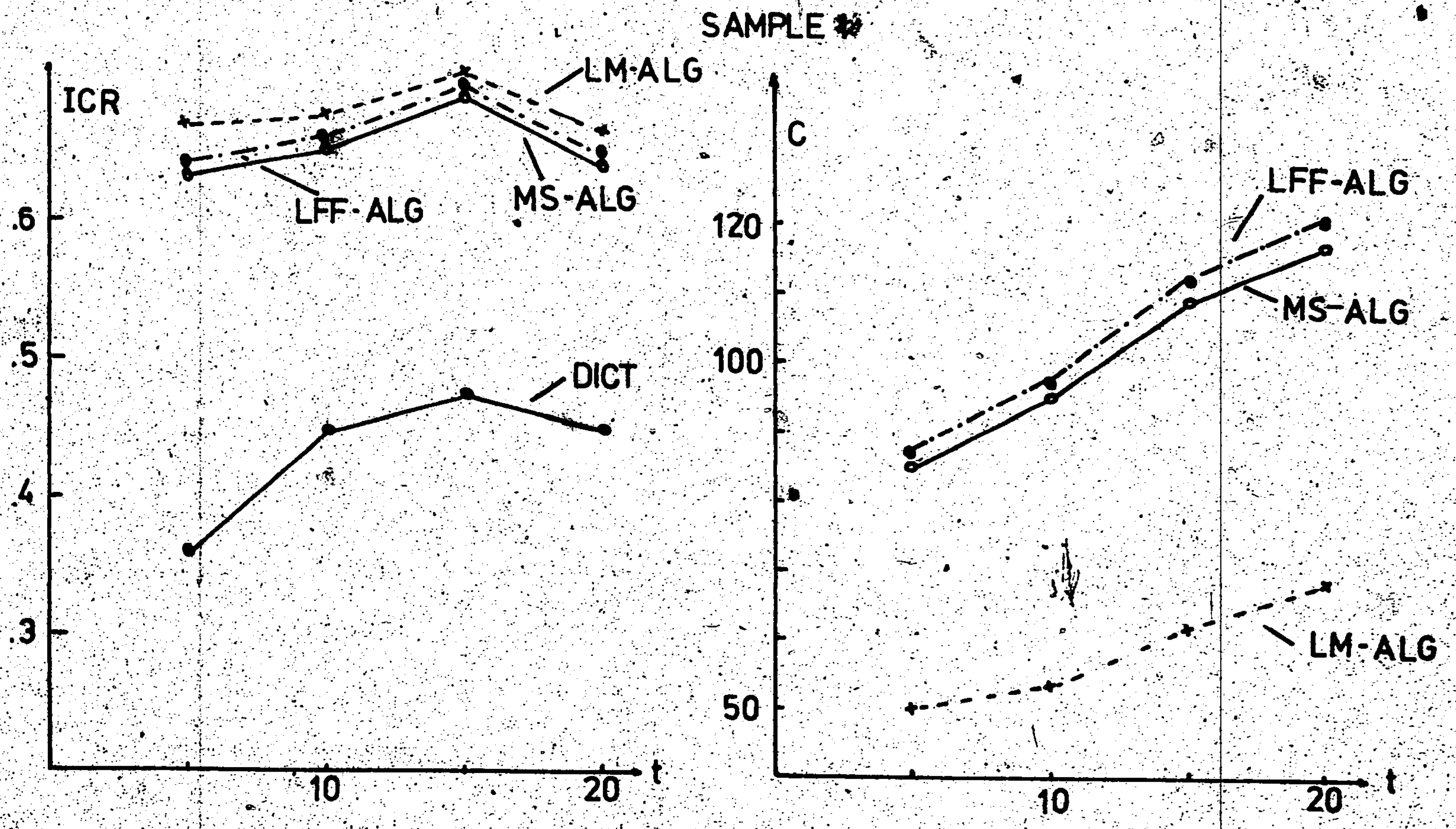


Figure 30

Compression and Coding Time after Coding with Wordfragments

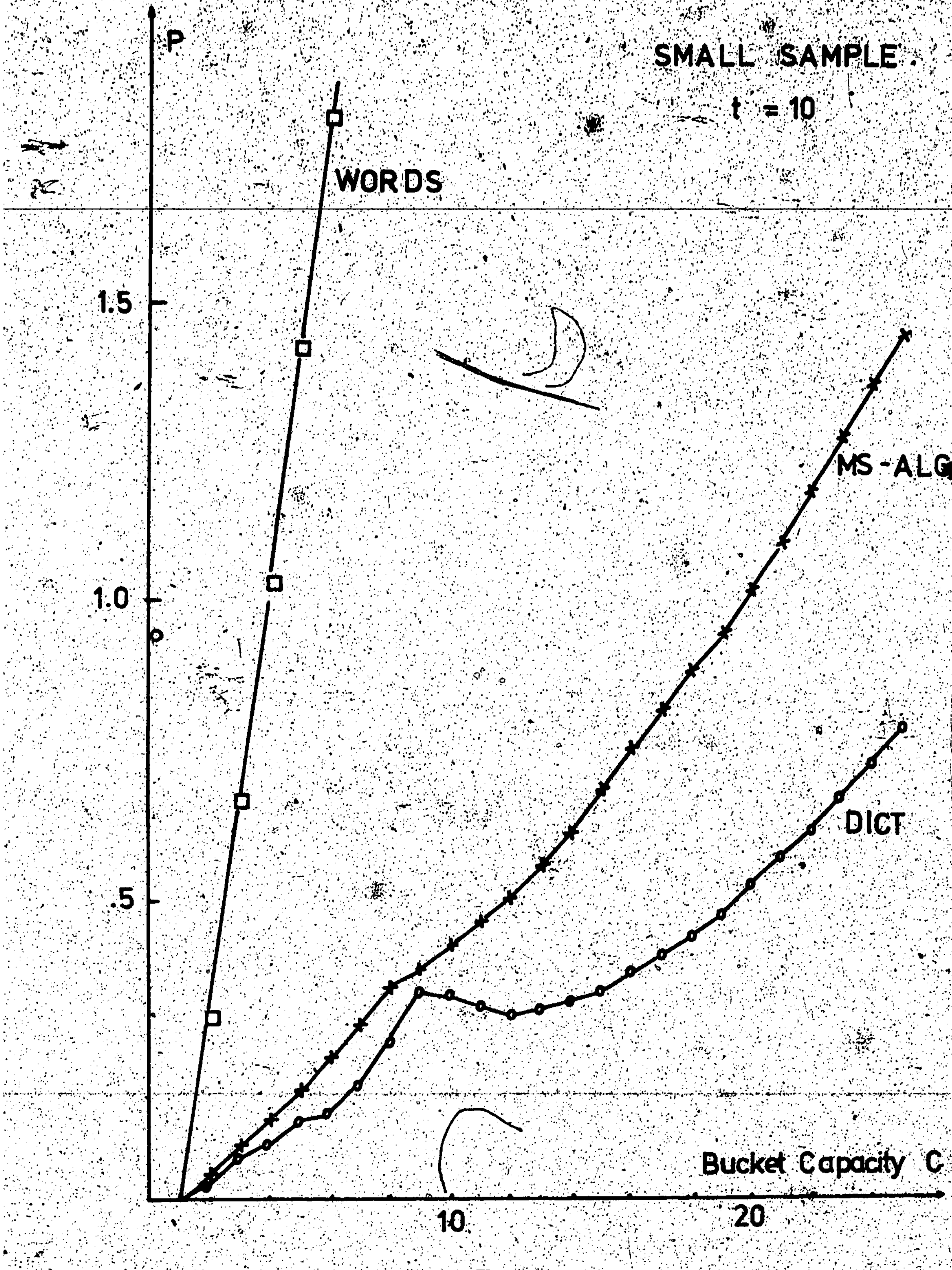


Figure 31
Memory Utilization Factor P after Coding with Wordfragments

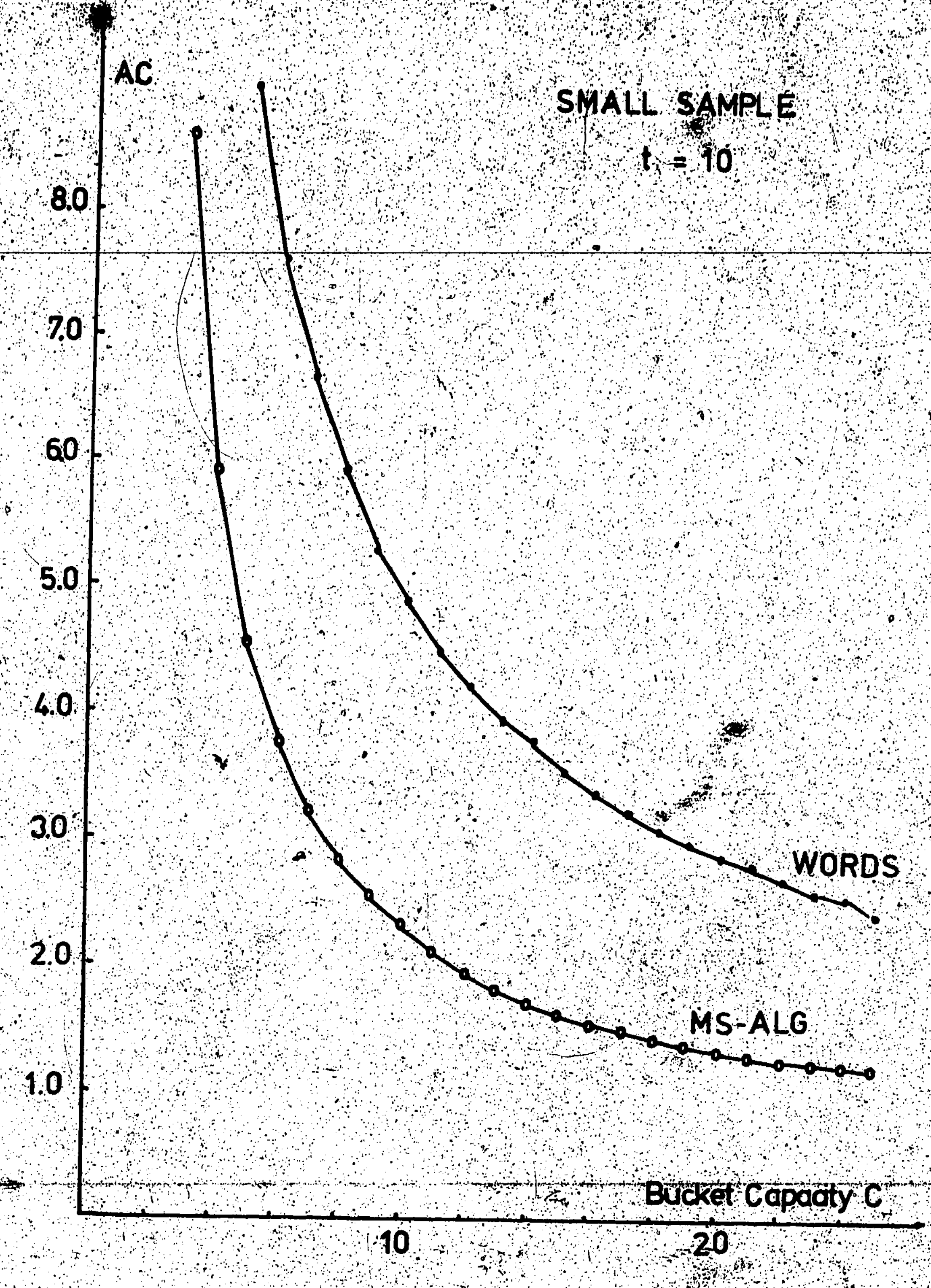


Figure 32
Average Number of Diskaccesses after Coding with Wordfragments

5.4.3 Word versus Text Fragments

With the results of word and text fragment coding now available, it is possible to discuss the merits of the two alternate language elements.

The differences in the compression obtained with the two different language elements are insignificant. This is especially true when it is remembered that text fragments of length up to 10 were used, versus a length of 8 for word fragments. Better compression may be achieved by an increase in the maximum length of word fragments in the dictionary. The longer word fragments could be easily found with the approach described by Hultgren and Larsson [34] without an increase in complexity.

A strong argument in favor of the use of word fragments is presented by consideration of the inverted index. The number of index entries for word and text fragments at the same threshold are about equal. Since text fragments contain a considerable number of fragments unsuitable for retrieval, the actual number of entries would be lower than for word fragments. This may result in reduced recall.

Furthermore the average number of disk accesses is about the same. Comparison of the memory utilization factors of the different fragment types shows that word fragments are more tightly packed.

This tips the scales in favor of word fragments; they seem to be the more natural choice. Colombo and Rush [19] have found word fragments suitable for retrieval in a sequential search system. For different reasons, following a somewhat different logic, it can be concluded that they are also suitable for retrospective searching with an inverted index.

The above comparison was only between word and text fragments and no comparison was made with words. Such a comparison can only be made after the retrieval behaviour of such a system has been investigated. This is attempted in the next chapter.

One interesting problem, however, remains still open. It is the question of the validity of the results with regard to other databases. Is the behaviour of word and text fragments with respect to selection and coding essentially the same for other databases ?

In the present treatment a comparison of different databases was omitted so that consideration could be given to the retrieval performance of a system that uses fragments. This was regarded as a necessary part of the present thesis.

CHAPTER VI

QUERY PROCESSING

This chapter is concerned with [illegible]lems that occur when processing a query in a system [illegible] fragments as indexing elements. The previous chap[illegible]wed that the use of fragments required significant changes in the system generation phase. It can be expected that the query processing is also affected. In the last chapter it was concluded that it is better to use word, rather than text, fragments. For this reason text fragments will not be considered in this chapter. However, one possible advantage of text fragments for query processing must be mentioned.

Retrospective retrieval systems that are based on words as language elements normally allow only single words to be used as search terms. Longer phrases must be expressed as two separate terms connected by the adjacency operator, which is first implemented as an "AND" operator. The documents retrieved by the manipulation of the appropriate index rows are then checked for adjacency by a sequential search before the hits are output to the user. If there exists an index fragment to connect the two terms in a phrase this checking may be unnecessary. The elimination of text fragments from further consideration thus removes the option of searching for phrases without having to perform a sequential search. If this option is desirable, because many users prefer phrases as search terms, it might be better to use text rather than word fragments. However, the present chapter will only investigate the retrieval behaviour of a system that uses word fragments.

6.1 Expansion of Search Terms

One major problem to be solved when processing a query is to find the fragments related to a search term. Each search term must be expanded into a variable number of fragments, but only index fragments are of importance. The tree search, which finds all fragments as part of the coding algorithms, may be used. However, it is insufficient for query processing, since it is incapable of handling truncated search terms.

A different approach is proposed in order to treat truncated and untruncated search terms and produce the desired fragments. However, two auxiliary tables must be constructed before the algorithm can operate. The relationship between the index, the dictionary and the auxiliary tables is shown in figure 33 and will be explained in the following sections.

Fragment Dictionary:

Assume the ND(t) dictionary fragments are stored on length as the primary key, and alphabetically within each group of the same length. Let P(i) be a pointer to the last fragment of the group of length i. The dictionary can thus be considered as a structured list and each element can be uniquely identified by its position in the ordered list. If DF represents the list of all dictionary fragments then the element DF(i) is well defined. Each dictionary fragment which also functions as an index fragment has an associated pointer to a row of the inverted index. If all fragments are used as index elements and fixed length buckets are used, these pointers can be omitted, because a straight forward calculation of the bucket address is possible.

Bigram Directory:

The main auxiliary table for query processing can be called a "bigram directory". It can be envisaged as an inverted index with the bigrams as the index elements and the dictionary fragments as the records being indexed. This directory is generated in the following manner.

All fragments of length two or greater are partitioned into their constituent bigrams. With each of these bigrams the following information is stored to form one record. The fragment index i in DF from where the bigram originated, the position of the first bigram letter in the fragment DF(i) and an indicator of whether the bigram was the last one in DF(i).

Assume that all B(t) different bigrams are put in an alphabetically ordered list BL, so that they can be indexed and the element BL(j) is well defined. The bigram directory BD can be represented by two matrices I and T whose elements are lists. Let p be the maximum fragment length. The size of I and T is B(t) x p. The elements of these matrices I and T can be defined as follows:

I(m,n) = List of all fragment indices which have bigram BL(m) as a nonterminal bigram at position n in the fragment.

T(m,n) = List of all fragment indices which have bigram BL(m) as a terminal bigram starting at position n in an n+1 letter fragment.

The bigram directory is a structure on a level above the dictionary and is essentially a rigidly structured collection of pointers into the dictionary. This structure can be used for right truncation but is incapable of handling left truncation. To handle left truncation efficiently another auxiliary table must be introduced.

<u>Terminal Letter Directory</u>

Assume that the L different letters in the alphabet are ordered according to a given collating sequence, so that the k-th element in the sequence is well defined. The terminal letter directory TLD is a vector with L components and each component is a list of indices of BL. The elements of the TLD are defined as follows:

TLD(k) = List of all bigram indices for which the associated bigram has letter k of the alphabet as the last letter.

This table allows rapid access to all bigrams which end in a given letter. Also, all bigrams starting with a certain letter can be found easily since they are located in adjacent positions of the bigram list BL. It may be convenient to set up an initial letter directory but is not a necessity.

These auxiliary tables enable the algorithm to link fragments and search terms. The truncation modes allowed for the search terms are those described in chapter three, namely unlimited truncation indicated by * and character truncation indicated by $. Simultaneous left and right truncation are admissible. The algorithm itself can be described with terminology borrowed from list processing, not unexpectedly, since the elements of the bigram and terminal letter directory are lists.

Let & denote the logical 'AND' operation performed on two lists and || the concatenation of one list to the other. The operator & has precedence over ||. Let E denote the empty list with the following properties:

$$A\&E = E \qquad A||E = A .$$

ALGORITHM:

Let S be a search term of length r represented by the letters L(1)...L(r). Break this term into its r-1 constituent bigrams L(i)L(i+1) with $1 \leq i \leq r-1$. Locate these r-1 bigrams in the bigram list BL and set k(i) (i=1,.. r-1) to the indices of BL where the bigrams are found in the list. If the j-th bigram cannot be found set k(j) to zero. Set m to the following value:

$$m = \begin{cases} p & \text{if } (r-1) > p \\ r-1 & \text{if } (r-1) \leq p . \end{cases}$$

Now construct two matrices I' and T' of size (r-1) x m. The elements of I' and T' are defined as follows:

$$I'(i,j) = I(k(i),j)$$

$$T'(i,j) = T(k(i),j)$$

for $1 \le i \le r-1$ and $1 \le j \le m$ and $k(i) \ne 0$. For values of $k(i)=0$ set for all $1 \le j \le m$

$$I'(i,j) = T'(i,j) = E .$$

The scheme to locate all fragments contained in a search term operates in the same manner for both full and truncated terms. The only difference is in the specification of boundary values of another matrix. Assume that the matrices I' and T' have been generated. Let Z be an (r+1) x (m+1) matrix of list elements.

1) <u>No Truncation</u> S = L(1) L(2) ..... L(r)

a) Boundary Values

Set $Z(r,j) = E$ for $1 \le j \le m+1$

b) Calculate the elements of the lower triangular matrix Z of size (r-1) x m. The elements Z(i,j) are given by the recursive formula

$$Z(i,j) = Z(i+1,j+1) \,\&\, I'(i,j) \;||\; T'(i,j) .$$

The fragment indices contained in $R = \,||\, Z(j,1)$ $(1 \le j \le r-1)$ are the desired fragments completely contained in the search term.

2) <u>Unlimited Right Truncation</u> S = L(1) L(2) ... L(r)*

a) Boundary Values

Set $Z(r,j) = \,||\, I'(k,j) \,||\, T'(k,j)$ for $1 \le j \le m$ and $Z(r,m+1) = E$. k is a variable index that may assume all values of an index set Q. This index set Q contains the indices of all the rows of the bigram directory for which the associated bigram starts with letter L(r).

b) Calculate the lower triangular matrix Z of size (r-1) x m using the recursion formula

$Z(i,j) = Z(i+1,j+1)\&I'(i,j)||T'(i,j)$.

Again $R = ||Z(j,1)$ $(1\leq j\leq r)$ contains all the indices of the fragment linked to the search term.

3) <u>Unlimited Left Truncation</u> S = * L(2).... L(r)

a) Boundary Values

Set $Z(r+1,j) = E$ for $1\leq j\leq m$

Set $Z(1,j) = ||I'(k,j)||T'(k,j)$ for $1\leq j\leq m$ and $Z(1,m+1) = E$. k is again a variable index assuming all the values of an index set G. The set G contains the indices of all the rows of the bigram directory for which the associated bigram ends in letter L(2). G is just the list of indices provided by the terminal letter directory corresponding to the letter L(2).

b) Calculate the full matrix Z of size r x m using the recursion formula:

$Z(i,j) = Z(i+1,j+1)\&I'(i,j)||T'(i,j)$

The desired fragment indices of the fragments completely contained in the truncated term are found in $R=||Z(j,1)$ for $2\leq j\leq r$. Those extending beyond the specified letters are found in $R' =||Z(1,j)$ for $1\leq j\leq r$.

It can be seen from these three cases that it is very easy to combine the two different boundary values for a term truncated on both sides.

The single character truncation can be handled by the same scheme but with modified boundary conditions. Assume that there are q single character truncation indicators on a side of the search term. When the boundary values are set in the algorithm it is necessary to prescreen the elements of Z which are used for initialization. References to all fragments which are incompatible in length with the specified truncation must be deleted. Using the length pointer P(i) into the dictionary fragment list DF allows this to be done without complication. In the case of right truncation include only those indices x of Z(r,i) for which $x \le P(k)$, with k defined as

$$k = \begin{cases} i+q & \text{if } i+q \le p \\ p & \text{if } i+q > p \,. \end{cases}$$

With left truncation eliminate those indices x of Z(1,i) which do not satisfy $x \le P(k)$ with k as defined above. This small additional effort enables the handling of both truncation modes and all fragments contained or partially overlapping into the search term are found.

Before the query can be processed the index fragments must be extracted. Since the index entries [illegible] by the coding algorithm when compressing the database, this algorithm must be applied to the set of fragments retrieved by the above scheme. The fragments of length greater than two that are selected by the application of the coding algorithm are the desired index fragments.

This approach is straightforward for untruncated terms, but complicated with truncation. The fragments can be divided into two sets, namely the set C containing all the fragments completely covered by the specified term, and the set A containing the fragments that overlap into the term. The coding algorithm must be applied more than once, each time operating with all the members of C and one element of A. If A has t members, then t groups of fragments will be generated. The index rows of the fragments in each group must be "and"ed to produce just one list of document numbers. The t lists that result from the t groups must be concatenated to produce the final list which represents the search term during further processing.

This term expansion is a time consuming process, but because of the one-shot nature of retrospective questions it is not too serious. An example of an expanded question using the word fragment dictionary with a threshold of 10 can be found in Table 15. The example shows how this expansion can lead to the retrieval of unwanted fragments. The errors in connection with truncated search terms can therefore be expected to be quite large.

```
QUE
    AND ([TIT,MAGNET*],
          [TIT, MATERIAL$] )
END
```

Possible subfragments

```
Term 1:
              C ={AG,AGE,ET,NET}

              A ={ETC,ETE,ETH,ETT,ETTE}

Term 2:
              C ={MATE,TE,ATER,IAL,RIAL}

              A ={ALS,ALL,ALT}
```

LFF-Algorithm used to expand Question terms

```
QUE
   AND
    [TIT,    M|AG|NET
             M|AGE|N|ETC
             M|AGE|N|ETE
             M|AGE|N|ETH
             M|AGE|N|ETT
             M|AGE|N|ETTE ],
    [TIT,    MATE|RIAL
             MATE|R|I|ALS
             MATE|R|I|ALL
             MATE|R|I|ALT ]
END
```

Table ~~15~~

EXPANDED QUESTION

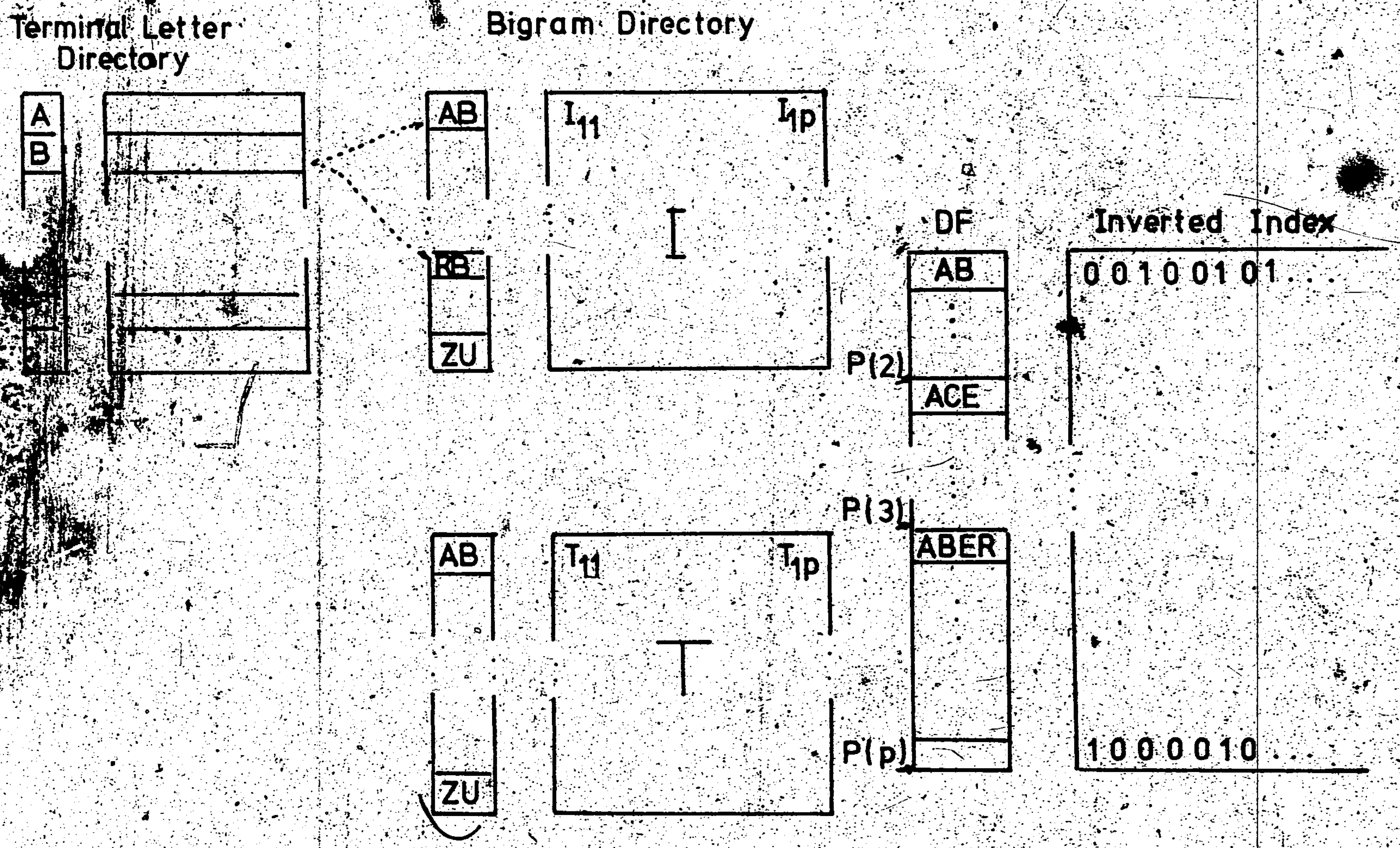


Figure 33
Auxiliary Tables for Query Processing

6.2 Treatment of Logical Operators

The scheme described previously solves the problem of expanding the question terms, and the last open problem involves the handling of logical operators in a query. These operators can be divided into three groups, the unary NOT, the two standard operators OR and AND and the powerful ADJ, PRE and WITH. The unary operator NOT has a special significance for a fragment system and will be discussed first.

NOT: The implementation of NOT in a word oriented retrieval system based on an inverted index is straightforward. The list A of document references corresponding to the negated term is retrieved and compared with the list B already obtained from the rest of the query. Document numbers common to A and B are deleted from B and the list B' obtained is used for further processing.

In a fragment system the NOT causes some unexpected problems. A document may contain a fragment of the negated search term as a part of a different word. This means that an entry in the inverted index for this document exists. The document could satisfy the question logic, but would be deleted if the same procedure as in a word oriented system would be applied. The chance of such a happening is quite high since one particular fragment can be the subfragment of many different words. The recall of the system is likely to be lowered significantly by this phenomenon.

The solution to this problem is to ignore the NOT operator completely on the index level and postpone its effect until a document is decoded. The inverted index yields a list of possible hit candidates since a query must contain at least one unnegated term. After these "candidate" documents are decoded a sequential search can check for the delayed NOT term and inhibit the output of documents containing this term.

AND-OR: These two common operators can be handled in the same manner as in a word oriented system. If two search terms are connected by an AND operator the two lists of document references are retrieved from the index. These lists are compared and only numbers common to both lists are put on the reduced list representing the result of the AND. The list of document numbers resulting from an OR operation is simply the concatenation of the two index lists corresponding to the two terms. This appended list may contain duplicates which should be eliminated before further processing takes place.

ADJ-PRE-WITH: It has been mentioned in Chapter three that in a retrospective system based on an inverted index these three operators must be implemented in two stages. The first stage is to treat all three as an AND. The strong requirement of a prescribed ordering of terms in the document is impossible to implement with the inverted file since this information is not stored there. The extra check must be performed in the second stage, on the document level by a sequential search. A fragment based retrieval system can follow the same approach. Replace any of the three operators by AND and process the query. Retrieve and decode all the documents satisfying this "weaker" logic. Perform a sequential search for the full logic on the decoded documents and discard all those that do not satisfy the conditions imposed by ADJ, PRE and WITH.

From the above considerations it can be seen that the adjustments necessary to the search logic are minor. However, a sequential search on the documents retrieved by the index is unavoidable for all queries that contain NOT, ADJ, PRE and WITH logic. A sequential search is also beneficial for queries containing only AND and OR operators, since it will eliminate false hits arising from the occurrence of a document that contains two fragments that are part of words different from the search term. To assure an error free operation with the desired operators the use of an inverted index with a fragment retrieval system must be followed by a sequential search.

6.3 Experimental Results

Precision and recall are two standard measures for evaluating the performance of a retrieval system. A set of specimen questions is then required. Furthermore a considerable amount of manual searching of the data base is involved in the calculation of recall. It was decided not to create an artificial set of test questions for the following reason. It would have opened the way for the criticism that any method may be made to look good by choice of a suitable example. In addition, the aim of the present study is not to evaluate a particular system's performance, but to make a comparison between word and fragment oriented retrieval systems. These reasons were considered sufficient to justify discarding the use of precision and recall. The approach taken was as described below.

The parameters selected for the comparison are ones which directly affect the system's user. The first parameter of interest is the number MI of words missed by use of a fragment inverted index. All words which are concatenations of single letters cannot be retrieved since inverted index entries exist only for fragments of length greater than one. The parameter MI is independent of the algorithm which makes the index entries. All three algorithms investigated will find a fragment of length greater than one in a word if it exists at all. During retrieval a question term which is represented by single letters only must be ignored completely. This will result in lower recall and is quite noticeable if the term is used with OR-logic but is negligible with AND-logic, unless all terms in the concept have no index entries.

Obviously the quantity MI of missed words is a function of the threshold t. Values of MI as a function of t are given in Table 16.

The second parameter of interest is the number of false hits originating from the operation of the inverted index alone. The user should not see these false hits because they should be suppressed by the successive sequential search. However, the number of false hits is still of great interest to the system designer, since the number of documents to be scanned sequentially should be kept small in order to provide good response time.

In contrast to MI the false hits are a function of the coding algorithm. However, since the number of index entries differs very little for the three algorithms under discussion, any one of the algorithms can be used to give an idea of the magnitude of the errors to be expected. To determine the false hits an inverted index based on word fragments was generated, and the hits retrieved for each search term with the fragment index were compared with those retrieved using the original word index. Index entries were generated by the LFF-algorithm.

The experiment was performed only with sample one. Author, title and subject heading field were considered (to form one record. During retrieval each search term was broken up into its index fragments as selected by the LFF-algorithm. The associated fragment index rows were retrieved for all the index fragments and an AND operation executed. The resulting list of document numbers was compared with the index row of the original word. Statistics were collected during this phase to determine the influence of the threshold.

The statistics gathered during the generation of the fragment index and the retrieval of single terms are presented in table 17. They require some explanation.

It is quite common to find that one index fragment appears more than once in a record since each record consists of author, title and subject heading. This is a feature peculiar to the MARC tapes since most of the subject headings repeat one or more words already present in the title. These duplicate index entries can be eliminated to save storage. This saving was predicted by the reduction factor of equation 5.2(5). The experimental results can now be compared with the theoretical prediction. The experimental values are .86, .85, .85 and .84 versus .97, .96, .91 and .89 calculated from 5.2(5). This shows that the independence assumption made in the derivation of 5.2(5) is unrealistic for the particular data base studied.

The main portion of the table displays the results of single term queries. For all four thresholds it gives the average number of false hits FH, and the fraction of queries

DFH with at least one false hit, as a function of the search term length. A graphical illustration of the tabled results can be seen in figures 34 and 35. It is sufficient to plot only the lowest and highest threshold because the other two graphs fit in between the ones plotted.

When the fraction of all queries with at least one false hit is calculated, it is seen to be quite high, ranging from 33 to 55 percent. An analysis of this error is attempted below in order to determine the main parameters responsible for the error and to find the dependency of the error on sample size. Actual and estimated error can be compared to see if any assumptions made were valid.

Let the data base have NR records with an average length of AR characters per record. The fragment dictionary to be used has ND elements with an average length of AF. For each index fragment there exists an associated index row with F entries. Assuming independence between individual records, the probability that one particular record number occurs in an index row is

$$pr = F/NR \; . \qquad 6.3(1)$$

It must be remembered that equation 3.3(7) links F to the total data base length TC which is also given by

$$TC = AR * NR \; . \qquad 6.3(2)$$

Substituting 3.3(7) and 6.3(2) into 6.3(1) yields

$$pr = (1/ND)(AR/AF) \; , \qquad 6.3(3)$$

which is independent of the length of the data base. Since AF and ND were found to be empirical functions of t, pr can be expressed as a function of the threshold, giving

$$pr = (AR/N0*A0))*[illegible]**(a+[illegible]) . \qquad 6.3(4)$$

Let p(k) be the probability that a word is a concatenation of k index fragments. Each word containing only one index fragment causes a false hit, but the probability for a false hit if the word is made up of k (k≥2) index fragments is given by pr**k. This is based on the assumption that a record may be regarded as made up by independent draws from the fragment dictionary. The contribution of each group consisting of k index fragments to the total error can now be given as

$$c(k) = \begin{cases} 1 & \text{if } k = 1 \\ pr**k & \text{if } k \geq 2, \end{cases}$$

and the total error ER is

$$Er = \sum_{k=1}^{kmax} p(k)*c(k) . \qquad 6.3(5)$$

Unfortunately this expression involves quantities p(k) which must be found experimentally, since they are dependent on the dictionary and the coding algorithm. Figure 3 shows the values of p(k) for sample one based on the LFF-algorithm.

It can be seen that the probabilities are almost independent of the threshold. The explanation is that with a low threshold there are about the same number of index fragments per word but they are longer than for a high threshold.

When the error is calculated according to 6.3(5), with the experimental probabilities p(K), calculated values of 29,19,17 and 16 percent are obtained. This is quite different from the experimental values of 33,40,45 and 50. The reason for this discrepancy is the assumption that records are obtained by independent draws from the dictionary.

Such an assumption is totally unrealistic since there are strong bonds between the fragments that constitute a word. The occurrence of one word causes a co-occurrence of these fragments.

Another factor, probably the main cause for the increased error, is the presence of the same index fragments in the plural and singular of the same word. The terminal letter s is usually a single letter fragment which has no effect in retrieval because it does not discriminate like an index fragment. Furthermore there are many words that have the same index fragments but in different order or with an intermediate single letter. All these effects accumulate and tend to increase the factor c(k) above the theoretical value of pr**k. The actual values of c(k) obtained in the experiment for a threshold of 10 are plotted in figure 37 together with the estimated c(k).

An analysis of the factors that contribute to the increase of the c(k)'s is beyond the scope of this thesis since it would involve mainly linguistic aspects. Some of the problems to be encountered occur also in connection with stemming algorithms [42] and, beside their obvious vocabulary dependency, are of a combinatorial nature.

Following the study of single term queries, it was decided to study the effect of question logic on the occurrence of false hits. AND logic was chosen and implemented in the following manner.

A pair of words was selected at random and was treated as a two term query connected by AND. The query was answered twice, first using the fragment index and then for comparison using the word index. All hits found with the first index, but not the second, were recorded. Table 17 contains the two macro statistics of the experiment, the percentage of queries with at least one false hit, and the average number of false hits per query for the queries that have false hits. For each threshold 21,000 word pairs were selected. Figure 38 shows the percentage distribution of false hits against the length of the shorter term of the pair. It also displays the number of queries and relative frequencies of one or more false hits for a threshold of ten. It can be seen that queries containing a term of length three or four are the cause of most false hits (78%). Table 18 shows some sample queries and the false hits retrieved with the fragments responsible for the retrieval underlined. A manual inspection of all false hits suggests two reasons for the poor performance.

1) A large number of word pairs share a common subfragment which reduces the number of AND operations necessary between

the index rows. This is quite serious, since each AND reduces the chance for a false hit. If the common subfragment is the only one found in both terms then a large number of false hits is observed.

2) When a term in a query contained a sequence of digits, then many false hits were encountered. This is due to the fact that the dictionary contains few fragments of length greater than one that represent sequences of digits. These numerical fragments are usually short and occur in many documents, but are in most cases a part of a number with more digits. In most instances it is not possible to find a second index fragment for a numerical term because the sequence of digits appears with a low probability in the data base. This reason appears, however, to be a special feature peculiar to the MARC tapes.

These two observations immediately suggest improvements to the system design. First, the number of fragments contained in a word could be increased by modification of the selection algorithm. All bigrams whose frequencies exceed the threshold could be included in the dictionary. This would not only reduce the percentage of words that do not contain at least a bigram, but it would also alleviate the problems described above. The addition of extra bigrams to the dictionary would of course distort the equifrequency property, and this is the price to be paid for better precision and recall.

Secondly the errors arising from the retrieval of numerical information can be reduced by including all possible two digit numbers in the dictionary. This would add approximately a hundred elements to the dictionary but would make the retrieval of dates feasible.

As a side effect, the experiment has confirmed the need for a sequential search of the documents retrieved by the index. It can be expected that more complicated queries may yield no false hits, but the additional search effort must be undertaken in any event, in order to treat positional logic.

The major conclusion from these experiments is that it is desirable to use a low threshold because of the fewer retrieval errors that result. This is in perfect agreement with the results of section 5.2 in which lower thresholds produced the better compression.

A conflict, however, arises with the inverted index results. The storage utilization factor increases with decreasing thresholds and results in greater use of disk space. From this point of view a high threshold is desirable. These conflicting requirements must be resolved by the designer of the system based on his knowledge of the available system resources and the purpose of the system.

SAMPLE 1

Threshold	Misses	Percentage
5	17	.61
10	42	1.51
15	60	2.14
20	62	2.20

The word inverted index had 2782 rows with a total of 5567 entries *

* This was obtained by eliminating all words of length less than three and discarding 36 common words with a stop list.

TABLE 16

STATISTICS ON WORDS NOT CONTAINING AT LEAST A BIGRAM

SAMPLE 1

Threshold	5		10		15		20	
Length	FH	DFH	FH	DFH	FH	DFH	FH	DFH
3	11.4	97.8	18.7	100.	23.0	100.	26.8	100.
4	3.9	73.5	8.6	82.8	10.6	84.7	12.5	90.[illegible]
5	1.6	52.1	2.9	57.3	4.4	70.6	4.8	76.6
6	.75	31.3	1.3	43.8	1.7	50.3	2.2	59.4
7	.43	20.4	.70	31.9	.84	36.4	1.1	44.9
8	.56	20.0	.70	22.2	.76	29.6	.89	33.2
9	.40	14.1	.50	16.4	.56	21.2	.60	[illegible]
10	.14	10.3	.21	14.6	.24	16.2	.28	20.0
11	.25	14.5	.25	13.[illegible]	.32	18.5	.31	20.2
12	.11	[illegible]	.17	9.85	.21	9.9	.15	11.3
13	.20	9.1	.20	11.3	.11	9.1	.14	11.4
14	.19	9.5	.14	9.5	.10	4.8	.19	9.5
15	.6[illegible]	12.5	.13	12.5	.13	12.5	.13	12.5
16	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0

Percentage of Words affected	33.4	39.3	44.7	50.3
Number of Duplicates	1401	1726	1851	2067
Average Number of Duplicates	1.03	2.18	2.78	4.56
Percentage of Added pairs with false hits	1.33	2.93	3.64	4.37
Average number of false hits per pair	1.18	1.37	1.45	1.47

Table 17

SUMMARY OF RESULTS FROM RETRIEVAL EXPERIMENTS

AND(, ROMAN,ANTHROPOLOGY) [RO|MAN][AN|TH|RO|P|OLOGY]

False HIT:

Smith, Ronald Gregor

The whole <u>Man</u>; Studies in Christian <u>Anthropology</u>

AND(LEADER,ADDRESSES) [L|EADER][A|DRESSES]

False Hit:

Hoffman, M. David

<u>Leader</u>ship in a Changing World

(Social Sciences, A<u>dresses</u>, Essays, Lectures)

Cayce Edgar, Cayce Hugh Lynn

The Edgar Cayce R<u>eader</u>

(Psychical Research Ad<u>resses</u>, Essays, Lectures)

AND(ACT,HISTORIC) [AC|T] [HI|STORI|C]

False Hit:

Freund , Ida

The Study of Chemical Composition; An A<u>cc</u>ount of its

Method and <u>Histori</u>cal Development

Priestley, Joseph

<u>Histori</u>~~cal~~ <u>Ac</u>count of the Navigable Rivers,Canals

and Railways Throughout Great Britain

Table 18

SAMPLE QUERIES WITH FALSE HITS

c

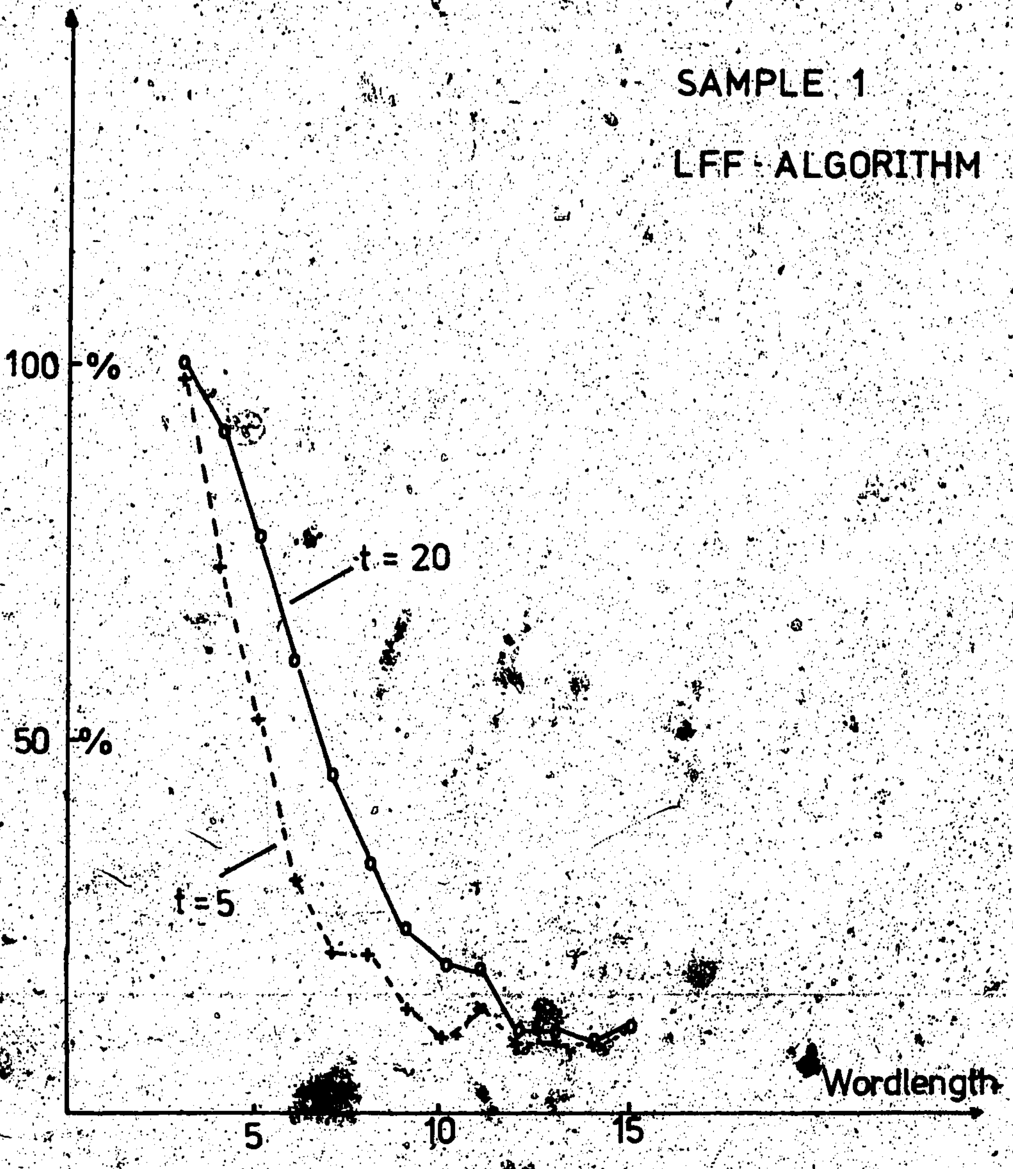


Figure 34

Percentage of different Words with false Hits

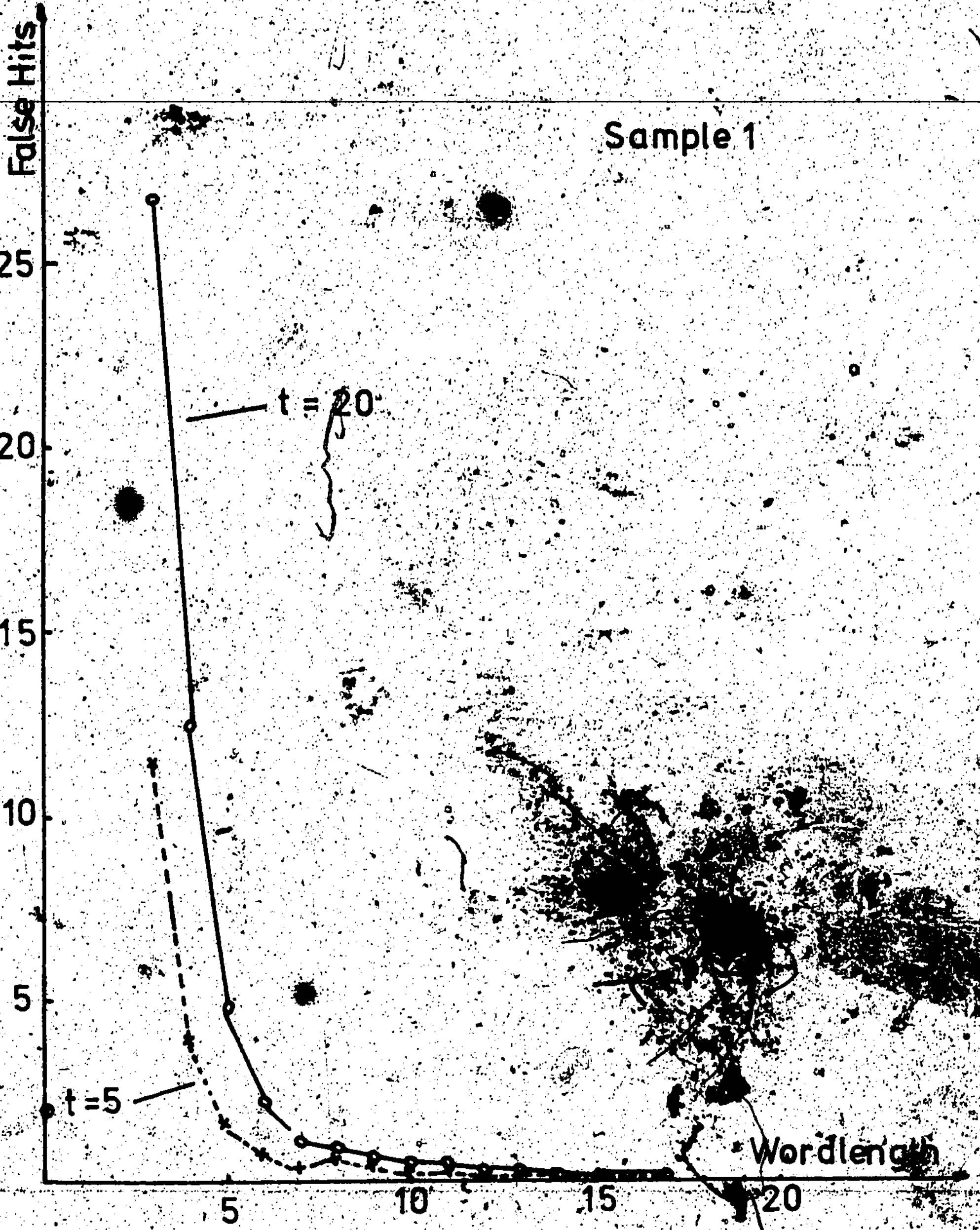


Figure 35

Average Number of False Hits for Single Term Queries

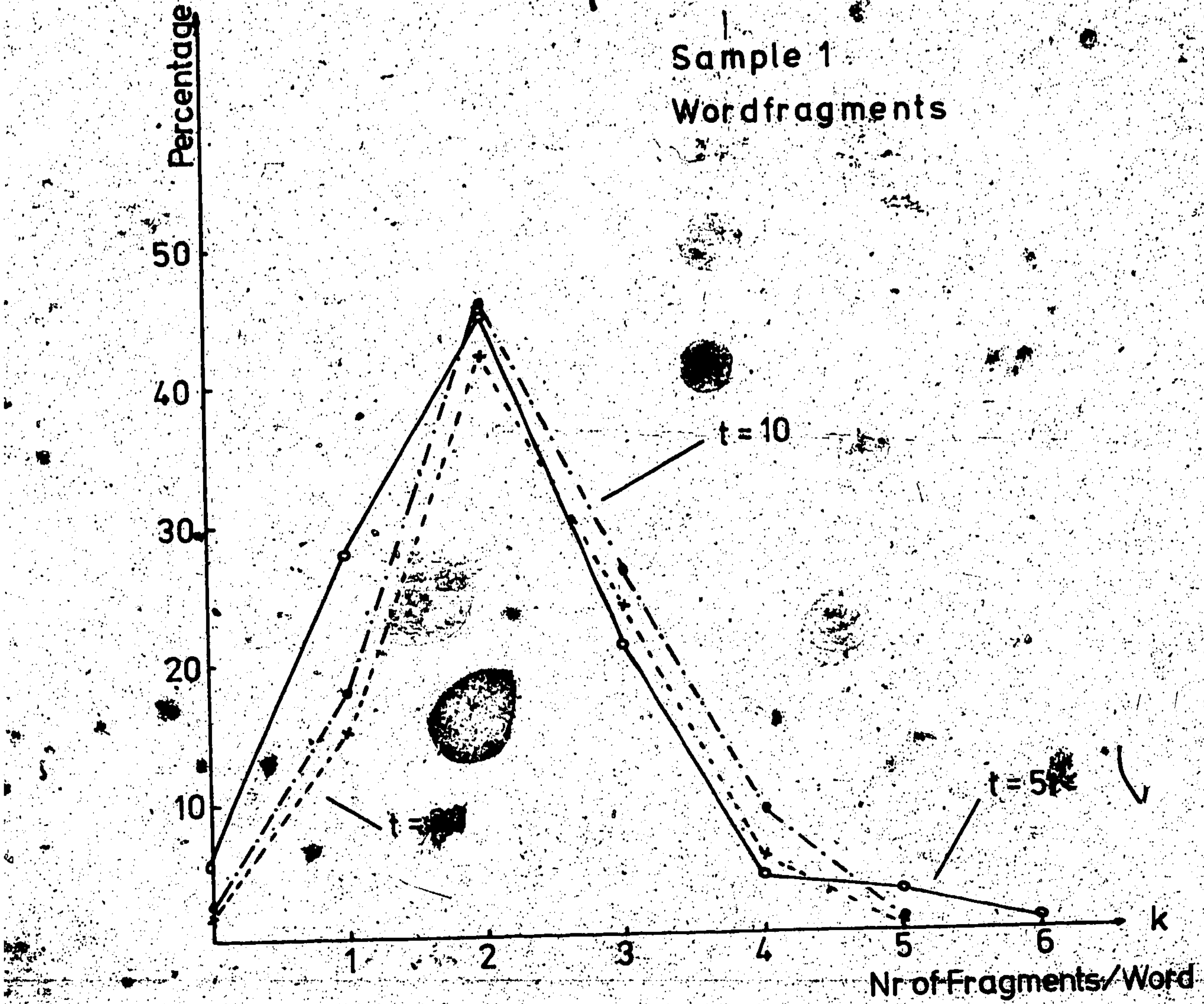


Figure 36

Percentage of Words with k Indexfragments per Word

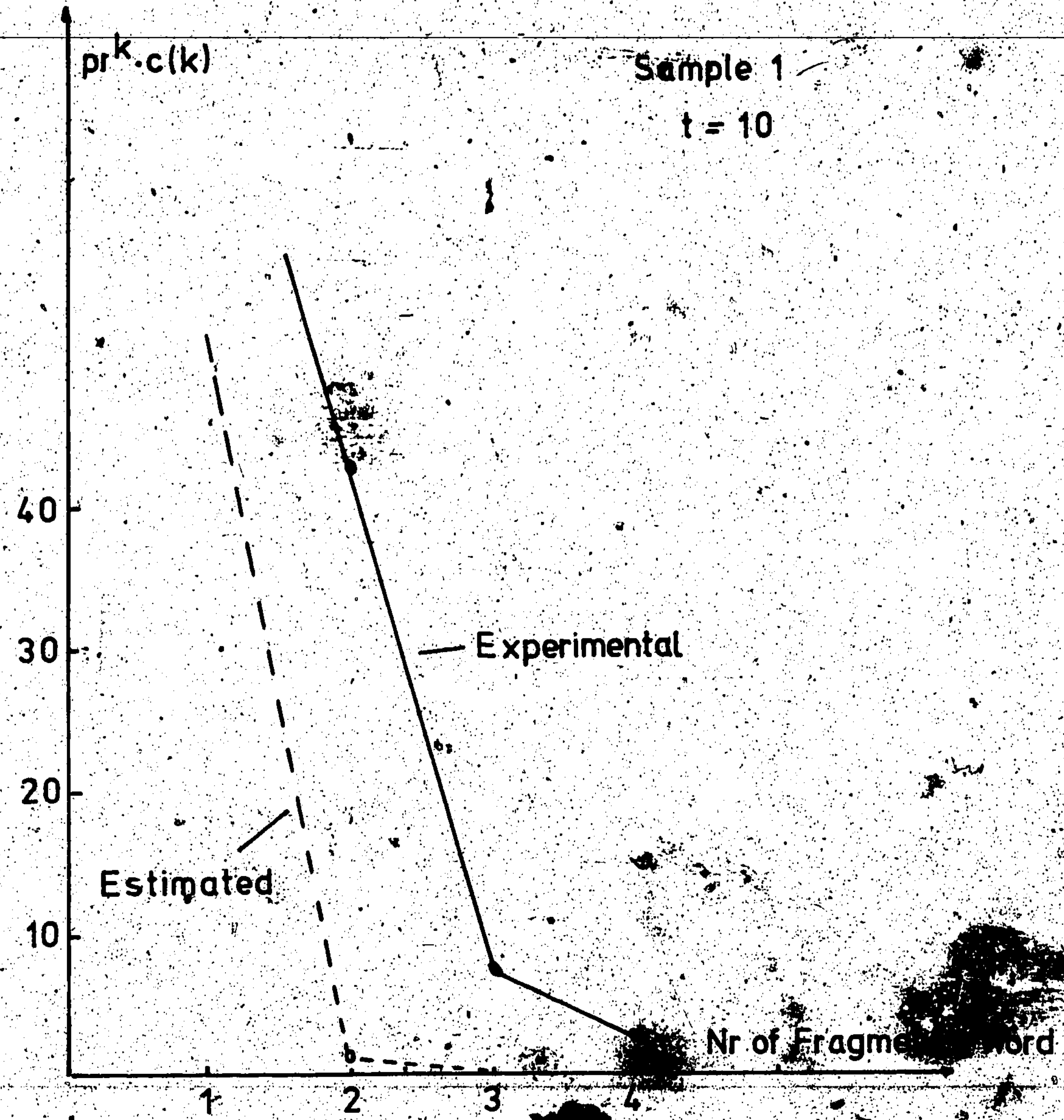
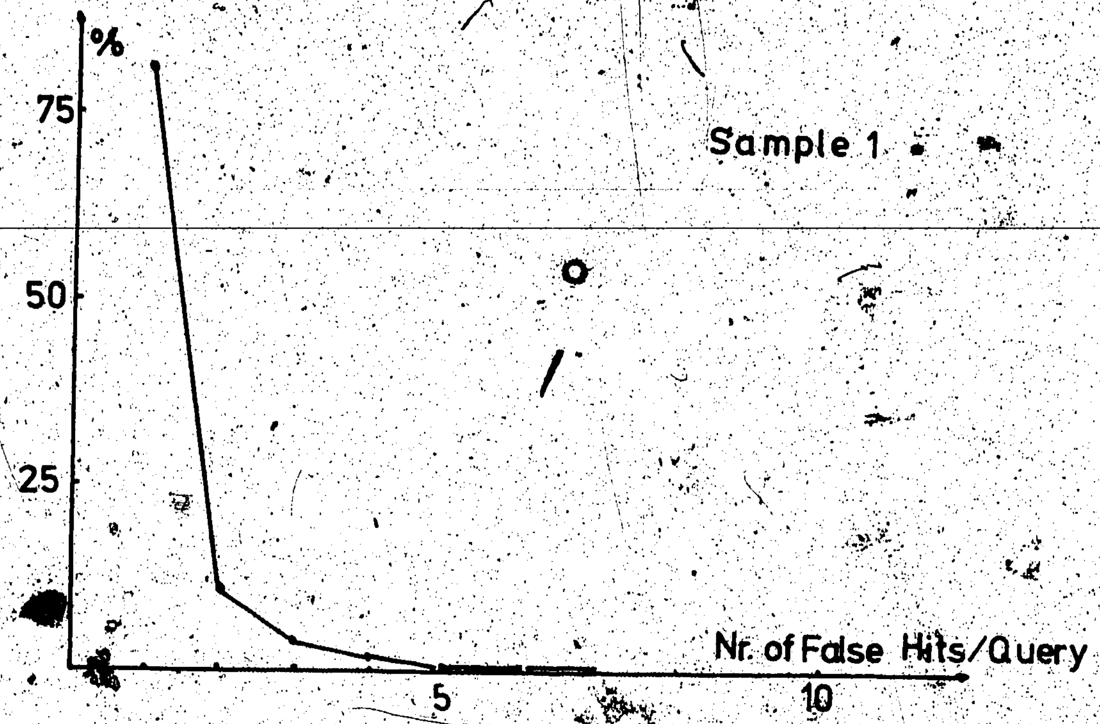
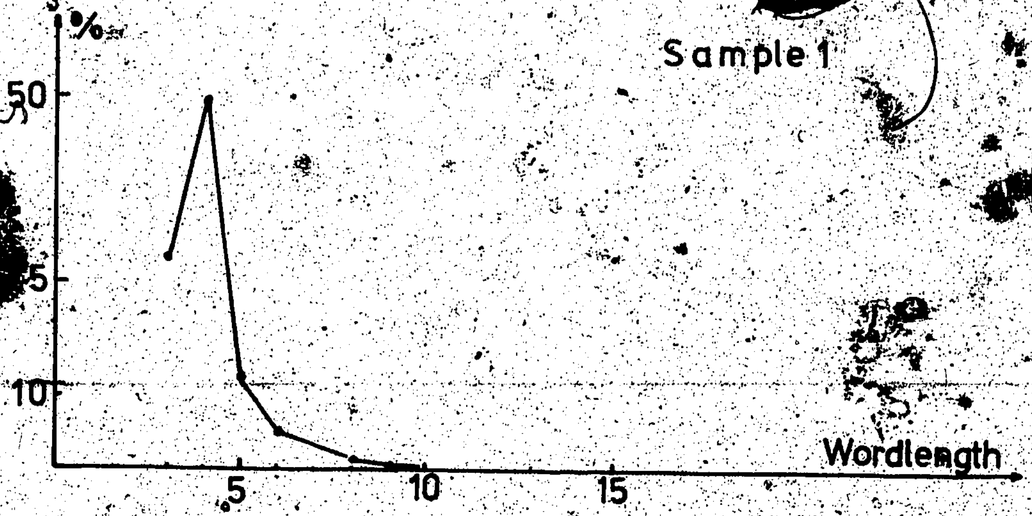


Figure 37

Experimental and Estimated Retrieval Errors



Percentage of 2 Term Queries with False Hits versus the Number of False Hits per Query



Distribution of False Hits for 2 Term Queries versus the Wordlength of the shorter Term

Figure 38

6.4 Guidelines for the Development of a Retrieval System based on Word Fragments

The previous sections do not provide any assistance and guidance to a system designer who wants to use word fragments in a retrieval system. It is not necessary to repeat the experiments described, if the subsequent considerations and steps are followed. The experimental evidence and the conclusions provided by the present thesis were used as a model to derive the following instructions. However, if the procedure is used, the designer must be aware of the inherent limitations.

The procedure was derived from two samples of the MARC tapes and can not be applied to other databases without restrictions. In these cases a repetition of the experiments described seems appropriate for a small database sample. The application of the procedure to the MARC tapes appears to be quite safe, since Lynch et al. [46] assured stability with regard to time. If a sample is used to determine the basic vocabulary parameters, such as average word length and repeat rate, the assumption about the homogeneity of the database mentioned in section 1.3 must hold true.

The sequence of considerations and calculations is quite dependent on the characteristics of the computer used to implement the system. Two different computer systems and Sample two will be used later on to exemplify the application of these considerations in the design of a fragment based retrieval system. The design steps are as follows and are applicable to the database or its representative sample.

1) Select the length k of [illegible] word which will represent the fragments in the compressed database. The codelength should be convenient for machine processing and compatible with the smallest addressable unit, for example the byte. It is highly desirable to have the capability of manipulating a codeword with a single machine instruction.

[illegible] the system uses word fragments, it must be [illegible]ed, that the code representing the fragments m[illegible]serve one bit for a flag, so that the last fragment of a word may be identified. Therefore k-1 bits are available for coding of the dictionary entries, and the dictionary is limited to ND = 2**(k-1) entries.

The two conditions, easy machine processing and reasonable dictionary size impose some limits on k. From the retrieval experiments it may be concluded that a value of k greater than nine seems to be sufficient to guarantee a small number of false hits. The experimental results of chapter four with regard to dictionary size suggest a value of k not greater than 16. Since most computers op[illegible] with either six or eight-bit-bytes, it may be conjectured, that a reasonable choice would be 12 bits. In eight-bit-byte machines this would necessitate half-byte manipulation, but in six-bit-byte machines no complication ar[illegible]es. However, in the case of a bit addressable machine like the Burrough's 1700, any k between 10 and 16 may be chosen.

2) Select the maximum length p for the fragments. Easy machine processing is again the dominating factor in the choice of this parameter.

3) Estimate the total number of fragment to[illegible]s TO(1) in the [illegible]tabase as p times the total number of non-blank characters in the database.

4) Use ND as a variable and obtain from figure 13 an absolute threshold frequency t(2). For a value of ND between 512 and 16,384, t(2) may be expected to range between 800 and 10.

5) Extract from Table 7 the sum of all fragment tokens up to length p and call it TO(2).

6) Calculate an estimate for the threshold frequency t(1) of the new database according to the re[illegible]ion

$$t(1) = t(2) * (TO(1)/TO(2))$$

This equation is based on the observation that the dictionaries for the two samples have approximately the same size for the same relative threshold frequency in the set of fragment tokens. The fraction t(2)/T0(2) represents the normalized relative threshold frequency. The absolute threshold for the sample is obtained by multiplication with the respective sample size.

7) Generate all fragments up to length p for the database and prepare a fragment frequency list.

8) Eliminate all fragments with a frequency below the threshold and apply the selection algorithm. Record dictionary size and average fragment length.

9) If the dictionary has more than ND elements increase the threshold and go back to step 8.

The selected set of dictionary fragments may be used satisfactorily if all of the following conditions are satisfied.

a) The size of the dictionary is sufficiently small that it can be held in primary memory during decoding.

b) The storage space needed for the list of accession numbers (associated with one index row) should be close to a multiple of the sector size of the disk.

c) Enough disk space must be available to store the inverted index. The total size of the index may be calculated from equation 3.3(4).

If condition (a) is violated, increase the threshold, if condition (c) is not satisfied, lower the the threshold. In the case of (b), the threshold adjustment may be made in either direction.

The dictionary determined by the above procedure should be satisfactory for the intended purpose. The above steps were laid out in such a form that the resulting dictionary will yield a small number of disk accesses and exhibit good retrieval behaviour.

To illustrate the above guidelines, a fragment oriented retrieval system was designed for two computer systems with contrasting characteristics. The design utilized in both cases the same database, namely Sample 2, which represented the accumulated MARC tapes for 1969.

It seems appropriate at this point to recall the dimensions of a word oriented system for Sample two. The dictionary contains approximately 32,000 entries with a total of 260,000 characters. The length of the compressed database using restricted variable length codes is about $5.5\ 10^6$ bits. The inverted index consists of 425,000 entries. 17 bits are needed to represent one entry and the total storage requirement for the index is $7.2.\ 10^6$ bits.

The two computer systems under consideration are assumed to consist of central processor, printer, card reader and primary memory. At least one tape drive must be available to allow the storage of the compressed database. Furthermore, at least one magnetic disk must be present to serve as random access storage for the inverted index. It must be assumed, that in case of need, all system resources may be available to the retrieval system. The system may be operational on the computer assumed, even though the generation of the index or the compressed database may exceed the system's resources. In this case it must be assumed that the auxiliary-files were prepared on a larger system and then transferred to the system under consideration. However, the problem of generating the system will not be dealt with, since it introduces aspects which only obscure the problem of the design.

The first machine to be used as an example is a CDC 6400 with a 60 bit word and a six-bit-byte. Low level language instructions are available to manipulate bytes within one word, since a character is also represented by a byte. The support disk of the system is assumed to be a CDC 853, where the smallest addressable unit, called 'PRU' by CDC, consists of 64 words. The maximum number of words to be transferred in one read is 4096 (64 PRU's), a restriction imposed by the pheripheral processor.

In this system the obvious choice for the length of a codeword is 12 bits, which according to step 1 of the guidelines implies a dictionary of 2048 elements. According to step 2 a suitable maximum fragment length must be selected. In this case a decision is easy, since p=10 is a natural choice. With this choice of p, one fragment may be stored in one word, and the total dictionary may be limited to 2k of primary memory. If it is desirable, say for decoding purposes, to prefix each fragment by its respective length, one byte could be used to store the length as a binary number. The maximum fragment length would thus be limited to nine. Using ND=2048 as a variable in figure 13, t(2) may be [illegible] as about 180. In [illegible] case of Sample two this value represents the absolute threshold frequency to be used in the selection algorithm. If the size of the dictionary resulting [illegible] the application of the selection algorithm is less than 2[illegible], the threshold can be lowered and the selection should be repeated. This process may be iterated until the dictionary size is less than 2048, [illegible] any further decrease [illegible] the threshold would result in a dictionary with more [illegible] elements. The average fragment length in the dictionary [illegible] then be determined, and by using equation 3.3(7) the average number of index entries per row may be calculated. However, the average fragment length may be estimated from figure 14, without the selection performed, since the approximate threshold mentioned above can be used as a variable. In this case some tolerance for possible inaccuracies in the prediction seems appropriate.

For the example the estimated average number of entries per row is about 270. Assuming that the list of document numbers is stored as one long bitstring, 4590 bits are needed. Two PRU's are sufficient to store this row and therefore the index requires a total of 4096 PRU's or 12.5 10^6 bits. The resulting disk utilization factor is .67, which implies that almost twice as much disk space is allocated than actually needed.

The second machine to serve as an example is a medium sized computer, a PDP 11 with a 16 bit word and an eight-bit, byte. Low level language instructions are available to manipulate bytes. However, if bits strings of length other than eigth are to be processed, special macro's must be implemented, since single instructions are insufficient. The disk connected to the system is assumed to be an RK05 with a capacity of 1.2 million words. The smallest addressable unit, the sector, contains 256 words. The maximum number of words which may be transferred in one access is limited to 65K words.

The choice for the length of the codeword is not as simple as in the first example. 16 bits are too many, since the storage of 32,768 dictionary elemts along with a useful program would exceed the maximum storage capacity of the machine. Again 12 bits seem to be the most suitable choice, since 12 bit codewords would be easier to process than a bitstring of other length. The system parameters would be similar to those of the first example. For this reason k was chosen as 10 to provide a contrast to the first example. A value of 10, however, implies cumbersome processing of codewords, an aspect which is important in the design, but is neglected here for the sake of contrast.

With k selected as 10, the dictionary may contain a maximum of 512 elements. The choice for the maximum length for the fragments is easy; any multiple of two will be satisfactory since a word contains two bytes and can be manipulated by a single instruction. Following the guidelines above, the absolute threshold may be extrapolated [illegible]820. Similar to the first [illegible] entries per row can be estimated [illegible] to [illegible] index row in the same arrange[illegible] as [illegible] requires 18,790 bits, which may [illegible] stored in fo[illegible]rs. The total index space is 2048 sectors or 8.4 10^{6} bi[illegible] storage utilization factor of .10. The inver[illegible] occupies [illegible] 45% of the space available on a single disk. The compressed database has a length of 8.7 10^{8} bits and may be stored on one tape reel.

Despite their different characteristics, the two systems have one thing in common, namely the average number of disk [illegible]. This may be attributed to the fact, that in both cases [illegible] index row may be transferred in one access. Both systems operate with a dictionary which is quite small compared with that of the word oriented system.

The major difference between the two systems occurs in the disk utilization factor, which is quite small in the second system, thus indicating very little wasted disk space. This behaviour may be contributed to the higher threshold used in the design of the second system, a phenomenon predicted by the previous experiments of this thesis. This may be verified by figures 17 and 19. The experiments also predicted, that the post index sequential search for system one would be shorter than in the second system. The second system would have to eliminate more false hits by this search and thus would have a slower response than system one.

The variation in the size of the inverted [illegible] compressed databases between the two systems is not crucial. The increase in size over the word [illegible] is quite reasonable considering the fact that the dictionary was eliminated. All these arguments [illegible] easy machine processing dominates the arguments for the choice of [illegible]. System two is hampered by the processing of codewords [illegible] which is somewhat incompatible with the machine word length. The best choice, even for system two, would be 12 bits.

From the above guidelines and supported by the examples it may be conjectured, that 12 bits is indeed the most suitable code length. A codeword of this length may be conveniently processed in a machine with a six-bit-byte, and with slightly more effort in machines with an eight-bit-byte. This seems to apply to all fragment systems regardless of the database or the host computer involved.

Further generalizations about fragment systems especially with regard to other databases cannot be justified. Too little is known about the change in fragment dictionaries which may be caused by a variation in the repeat rate or other vocabulary constants. However, the experiments described in this thesis can be performed with other databases and the same information may be extracted. The similarities and differences between the results of the experiment and this study may be used to predict the behaviour of a retrieval system using the new database.

CHAPTER VII

CONCLUSIONS

7.1 Summary

During the theoretical investigations and experiments described in chapters four, five and six many results were encountered. Some will be restated for clarity and continuity in the present section. This review is not intended to be exhaustive, but rather to present a short summary and overview of the main results.

In chapters two and three the theory underlying the use of fragments in retrospective systems was developed. Furthermore, equations were derived to predict the changes to be encountered when using fragments, rather than words, for retrieval and compression. Requirements of a suitable set of dictionary fragments were determined in chapter four; they were based on an attempt to avoid some of the known disadvantages of word oriented systems.

An algorithm was developed to select dictionary fragments from the set of all fragments contained in the database. This scheme utilized a single parameter, the threshold frequency, as the main variable. A knowledge of the inherent relationships among fragments was used in this algorithm, and suitable data structures for its implementation were described.

It turned out that the threshold frequency was the critical parameter that affected the results. The algorithm performed well with regard to the proposed measure of efficiency, and the experimental figures were in close agreement with those predicted by the equations of chapter three. The main findings of chapter four were the empirical equations that related entropy, dictionary size, and average fragment length to the threshold. These quantities were simple polynomial functions of the threshold and were found to hold for samples of different size. One of the constants in the equations was found to be almost independent of sample size. These empirical equations, if used in conjunction with the equations of chapter three, are considered to be of great value for the prediction of storage requirements of a fragment oriented system.

In chapter five the effect of fragment usage was studied with emphasis placed on the compressed database and the inverted index. It was seen that text fragments did not increase the size of the coded database by introduction of storage overhead. Word fragments, however, necessitated a special storage mode for the compressed records and thus introduced overhead. Three schemes were investigated and the one that used tagged codewords was found to be the one with the smallest overhead.

The inverted index was the least affected by the change of language elements. Fixed size buckets were found to be quite suitable because of the equifrequency property. It was observed that due to the limited size of the fragment dictionary substantial reductions in the storage for the index could be obtained by elimination of duplicate entries. [illegible] factor was calculated and found to be a function of dictionary size and the number of fragments per record. It was concluded that word fragments are more suitable than text fragments for use in retrospective systems.

However, the main concern was the algorithm which generates the compressed database and the inverted index entries. It has been shown that the problem of finding the minimum space form of the compressed database can be reduced to solving a shortest path problem for each individual record. Three different algorithms have been studied. The first is the MS-algorithm which leads to minimum storage use. The others are the LFF-algorithm and LM-algorithm.

The experimental results showed that there is very little difference between the compression ratios of the three algorithms although the MS performs the best and LM the worst. The time required to code the database was smallest with LM but largest with MS. Word fragments, inclusive of necessary overhead, produced compression ratios almost on a par with that of text fragments. The better suitability of word fragments to the retrieval phase gave strong support to the use of word fragments.

The expected conflict between the different requirements imposed on the dictionary by compression and retrieval was confirmed by the experiment. A small threshold was found to be best for compressing and to [illegible] in a smaller average number of disk accesses to the index. On the other hand use of a high threshold yields less compression and more disk accesses, but [illegible] index space [illegible].

After studying the effect of fragments on the system structure, attention was focused on the retrieval phase. It was necessary to develop a scheme which would retrieve all fragments linked to a search term and which would handle truncated terms. An algorithm with auxiliary tables based on list processing techniques was found to be suitable.

The logic operators in a query were studied next and it was found that the NOT operator could not be implemented in the same fashion as in a word oriented system. This complication, and the fact that some terms did not contain any index fragments, led to the conclusion that the documents retrieved by use of logic applied to the index must be searched sequentially with respect to the full logic. This was not considered to form an additional burden since the positional logical operators required this search in any event. One advantage of a post index sequential search was the elimination of any false hits caused by coincident occurrence of index fragments.

The experimental results of single term queries showed that a considerable fraction of words were affected by false hits, with the queries containing short terms being the most affected. The fraction of queries affected increased with the threshold, and provided further reason for the choice of a low threshold. A simple [illegible] fails to predict the error rate, even approximately, showing a serious deficiency of the model. The combinatorial contribution [illegible] error is small compared with the one caused by [illegible] effects. The complex linguistic relationships between words, word stems, plural and singular forms of a word have not been taken into account. When the effect of simple AND logic was studied the results were more encouraging, since the error was quite small.

All the experiments served to illustrate the conflicting demands for the choice of the threshold, but it was decided that the results alone were not sufficient to provide a general guideline for the choice. The selection of a suitable threshold must be left to the designer of a system who must take into account all system resources.

The main purpose of this study was the investigation of the feasibility of use of equifrequent fragments in retrospective retrieval systems. It has been shown that they are indeed practical, but require some changes in system structure and query processing. It can be concluded that a fragment oriented retrieval system will perform satisfactorily if two conditions are satisfied.

The two conditions are that a query should contain at least two search terms and each document retrieved should be subjected to a sequential search before hits are output to the user.

The first condition guarantees, more or less, the existence of at least one index fragment and [illegible] eliminate user inconvenience. If operated with these extra conditions the fragment system should perform as well as a word oriented system.

7.2 Suggestions for Further Research

The use of fragments in information retrieval is in the exploratory stage and the present study has treated only one of several aspects. Despite the limited scope of the present investigation it has led to several questions which are worthy of further investigation.

One of these problems, basically linguistic, is the study of database stability, which is equivalent to vocabulary stability for a specialized vocabulary. One such study, that of data base stability with regard to time, has been made by Lynch et al. [46]. Still open, however, is the question of how the vocabulary changes with different databases and how such a change affects the distribution of fragments as shown in figures 10 and 11. If these distributions can be explained theoretically, a substantial problem in itself, then what are the critical parameters of the database vocabulary that control the distribution?

The selection algorithm is itself the source of many new question. These questions mainly concern modifications to the algorithm to improve either compression or retrieval performance. Any such modifications should be treated through all the deliberations of chapters five and six. A possible modification to the selection algorithm is the inclusion of the most common words in the dictionary before selection takes place. Another modification is the inclusion of all bigrams, or a mixture of word and text fragments, in the dictionary in order to achieve better compression. An example could be the inclusion of text fragments like "AND_THE", "AND_OF" in the word fragment dictionary. Another approach that might be worthy of consideration is the inclusion of common word endings in the dictionary in order to improve retrieval.

Another interesting problem is the change in dictionary fragments caused by change of vocabularies. Can certain fragments be considered part of a "universal" fragment dictionary, which would be independent of the vocabulary or even of the source language?

One other aspect worth investigating is the use of fragments chosen solely to produce maximum compression. This probably implies the use of variable length codes for the fragments. What effect would this have on the inverted index and retrieval behaviour?

After determination of a selection algorithm which might incorporate some of the proposed modifications, the question of retrieval errors could be settled by a thorough linguistic and combinatorial analysis. This is a nontrivial problem and requires data that is not presently available. This data consists of statistics about retrospective queries, the length of the terms in the query, and the frequency of query terms in comparison to their occurrence in the database.

The last question arising from this study is fairly general and concerns the use of fragments in other areas of information retrieval. Rickman and Gardner [65] have shown that bigrams can be used for automatic indexing. Presumably there will be more, and other, applications for fragments. It is hoped that the present research has made a contribution towards the use of fragments in information retrieval and that it has served to provide guidelines for work on some of the remaining problems.

BIBLIOGRAPHY

1. Abramson, N., "Information Theory and Coding", McGraw Hill, New York 1963

2. Barton, I.J., Creasey, S.E., Lynch, M.F., Snell, M.J., "An information-theoretic approach to text searching in direct access systems", Private Communication, M.F. Lynch

3. Becker, J., Hayes, R., "Information Storage and Retrieval: Tools, Elements, Theories", John Wiley, New York 1963

4. Benjamin, R.I., "A Generational Perspective of Information System Development", Communications of the ACM, Vol 15 (1972), pp 640-644

5. Booth, A.D., "A "law" of occurrences for words of low frequency", Information and Control, Vol 10 (1967), pp 386-393

6. Boothroyd, J., "Shortest Path between Start and End Node of a Network (Algorithm 22)", Computer Journal, Vol 10 (1967-68), pp 306-307

7. Bourne, C.P., Ford, J.E., "A Study of Methods for systematically abbreviating English Words and Names", Journal of the ACM, Vol 8 (1961), pp 538-552

8. Briandais, R., "File searching using variable length keys", Proceedings Western Joint Computer Conference 1959, pp 295-298

9. Brillouin, L., "Science and Information Theory", Academic Press, New York 1956

10. Bush, V., "As we may think", Atlantic Monthly July 1945 In "Perspectives on the Computer Revolution", Z. Pylyshyn, Editor, Prentice Hall 1970

11. Byrne, J.G., Mullarney, A., "A survey of Text Compression", Paper presented IV Cranfield International Conference on Mechanized Information Storage and Retrieval Systems, July 1973

12. Canning, R., "Turing Award Citation 1973", Communications of the ACM, Vol 16 (1973), p 653

13. Cardenas,A.F., "Evaluation and Selection of File Organization. A model and system.", <u>Communications of the ACM</u>, Vol 16(1973), pp 540-548

14. Carroll,J.B., "Word-Frequency Studies and the lognormal Distribution", <u>Proceedings Conference on Language and Language Behaviour (1966)</u>, [illegible]ale, Editor, Appleton Century Crofts, New York 1968

15. Carroll, J.B., "On Sampling from a Lognormal Model of Word-Frequency Distribution", In "<u>Computational Analysis of Present Day American English</u>" by Kucera,H. and Francis,W.N. Brown University Press, Providence,R.I. 1967

16. Chomsky,N., "Three Models for the Description of Language", <u>IRE Transactions on Information Theory</u>, Vol IT 2(1954), pp 112-128

17. Clare,A.C.,Cook,E.M.,Lynch,M.F. "The identification of variable length, equifrequent character strings in a natural language database", <u>Computer Journal</u>, Vol 15(1972), pp 259-262

18. Collmeyer,A.J.,Shemer,J.E., "Analysis of Retrieval Performance for Selected File Organization Techniques", <u>Proceedings Fall Joint Computer Conference 1970</u>, pp 201-210

19. Colombo,D.S.,Rush,J.E., "Use of Word Fragments in Computer based Retrieval Systems", <u>Journal of Chemical Documentation</u>, Vol 9(1969), pp 47-50

20. Cooper,W.S., "Expected Search Length: A single Measure of Retrieval Effectiveness based on the weak ordering Action of Retrieval Systems", <u>American Documentation</u>, Vol 19(1968), pp 30-41

21. Damerau,F.J., "Markov Models and Linguistic Theory. An experimental Study of a Model for English", <u>Janua Linguarum</u>, Mouton 1971

22. Davidson,L.D., "Theory of Adaptive Data Compression", In "<u>Advances in Communication Systems</u>", A.V. Balakrishnan, Editor, Academic Press, New York 1966

23. Dolby,J.L., "An Algorithm for variable length proper name compression", <u>Journal of Library Automation</u>, Vol 3(1970), pp 257-275

24. DeMaine,P.A.D.,Springer,G.K., "The COPAK compressor", In "<u>File Organization</u>", Selected Papers from FILE 68, Swets and Zeitlinger, Amsterdam 1969

25. Enerson,T.J., "Explanation of Pheriphonics Voicepac 2000", Private Communication, T.J. Enerson 1973

26. Gilbert,E.N.,Moore,E.F., "Variable Length Binary Encodings", Bell System Technical Journal, Vol 38(1959), pp 933-967

27. Good,I.J., "Statistics of Language", In "Encyclopedia of Linguistics, Information and Control", A.R. Meetham, Editor, Pergamon Press 1969, pp 567-581

28. Harary,F., "Status and Contrastatus", Sociometry, Vol 22(1959), pp 23-43

29. Heaps,H.S., "Storage Analysis of a Compression Coding for Document Databases", INFOR, Vol 10(1972), pp 47-61

30. Heaps,H.S., "Computational and Theoretical Aspects of Information Retrieval", Unpublished Lecture Notes CS 560, University of Alberta 1971

31. Herdan,G., "The Advanced Theory of Language as Choice and Chance", Springer Verlag, New York 1966

32. Hsiao,D.,Harary,F., "A formal system for Information Retrieval from Files", Communications of the ACM, Vol 13(1970), pp 67-73

33. Huffman,D.A., "A Method for Construction of Minimum Redundancy Codes", Proceedings IRE, Vol 40(1952), pp 1098-1101

34. Hultgren,J.,Larsson,R., "A method defining a limited set of character strings with maximal coverage of text", EPOS Project Report, June 1973 Royal Institute of Technology, Stockholm

35. Information Systems Office ,Library of Congress "MARC Manuals used by the Library of Congress", Information Science and Automation Division, American Library Association, Chicago 1970

36. Jamison,W.H.,Blatter,H.,Zonis,F.I., "Synthesis of unlimited vocabulary speech using a computer controlled channel vocoder", Proceedings 1972 Conference on Speech Communication and Processing, April 1972 Newton Mass., pp 423-426

37. Kent,A.K., "The Chemical Society Research Unit in Information Dissemination and Retrieval", Svensk Kemisk Tidskrift, Vol 80(1968), pp 39-47

38. Kohavi,Z., "Switching and Finite Automata Theory, McGraw Hill, New York 1970

39. Lefkovitz,D., "File Structures for On-line Systems", Spartan Books, New York 1969

40. Libetz,B.A.,Stangl,P.,Taylor,K.F., "Performance of Ruecking's word compression method when applied to machine retrieval from a library catalog.", Journal of Library Automation, Vol 2(1969), pp 266-271

41. Long,J.M.,Barnhard,H.J.,Levy,G.C., "Dictionary Buildup and Stability of Word Frequency in a specialized medical area.", American Documentation, Vol 18(1967), pp 21-25

42. Lovins,J., "Error Evaluation for Stemming Algorithms as Clustering Algorithms", Journal of ASIS, Vol 21(1971), pp 28-40

43. Lowe,T.C., "Direct access memory retrieval using truncated record names", Software Age, Vol 1(Sept 1967), pp 28-33

44. Lowe,T.C., "The influence of database characteristics and usage on direct access file organization", Journal of the ACM, Vol 15(1968), pp 535-548

45. Lowe,T.C., "Effectiveness of Retrieval Key Abbreviation Schemes", Journal of ASIS, Vol 22(1971), pp 374-381

46. Lynch,M.F.,Petrie,J.H.,Snell,M.J., "Analysis of the Microstructure of Titles in the INSPEC Database", Information Storage and Retrieval, Vol 9(1973), pp 331-337

47. Mandelbrot,B.M., "An informational Theory of the statistical structure of language", Proceedings Symposium on the Applications of Communications Theory, pp 486-490, W.Jackson, Editor, Butterworth, London 1953

48. Mandelbrot,B.M., "On the Theory of Word Frequencies and on Related Markovian Models of Discourse", American Mathematical Society Symposia on Applied Mathematics, Vol XII(1960), pp 711-715

49. Marron,B.A.,DeMaine,P.A.D., "Automatic Data Compression", Communications of the ACM, Vol 10(1967), pp 711-715

50. Martin,L.D., "A model for File Structure Determination for large On-line Data Files", In "File Organization", Selected Papers from FILE 68, Swets and Zeitlinger, Amsterdam 1969

51. Meadow,C.T., "The Analysis of Information Systems", John Wiley, New York 1967

52. Miller,G.A., "Some effects of intermittent silence", American Journal of Psychology, Vol 70 (1957), pp 31[illegible]313

53. Miller,G.A., Friedman,[illegible]man,E.B., "Length Frequency Statist[illegible]tten English", Information and Co[illegible]ol 1 (1958), pp 370-390

54. Mulford,J.B.,Riddell,R.K., "Data Compression Techniques for economic Processing of large commercial Files", ACM Symposium on Information Storage and Retrieval, 1971 Maryland, pp 207-215

55 Nicholson,T.A.J., "Finding the shortest route between two points in a network", Computer Journal, Vol 9 (1966-67), pp 275-280

56. Nugent,V.R.,Vegh,A., "Automatic Word Coding Techniques for Computer Language Processing", NTIS Technical Report AD 272401, 1962

57. Nugent,W.R.,"Compression Word Coding Techniques for Information Retrieval", Journal of Library Automation, Vol 1 (1968), pp 250-260

58. Onderisin,E.M., "The Least Common Bigram:A dictionary arrangement technique for computerized natural language processing", Proceedings 1971 ACM National Conference Chicago, pp 82-96

59. Ore,O., "Theory of Graphs", American Mathematical Society Publication Vol 38 (1962), Providence R.I.

60. Ore,O., "Graphs and Their Use", Random House, New York 1963

61. Peters,J.,Einfuehrung in die allgemeine Informationstheorie", Springer Verlag, Berlin 1967

62. Pollack,M.,Wiebenson,W., "Solutions to the Shortest Path Problem. A Review", Operations Research, Vol 8 (1960), pp 224-230

63. Ramamoorthy,C.V., "Document Compaction by Variable Length Encoding", Proceedings of ASIS, Vol 1(1964), pp 507-513

64. Reid W.,Heaps,H.S., "Compression of Data for of Library Automation", Automation in Libraries 1971, 2.1-2.2.7, Canadian Library Association Ottawa

65. Reza,F., "An Introduction to Information Theory", McGraw Hill, New York 1961

66. Rickman,J.T.,Gardner,H., "On-line Index-term Predictions using Bigram-Term Associations", Proceedings 1973 ACM National Conference Atlanta, pp 262-270

67. Ruecking,F.H., "Bibliographic Retrieval from Bibliographic Input. Hypothesis and Construction of a Test.", Journal of Library Automation, Vol 1(1968), pp 227-238

68. Ruth,S.R.,Villers,J.M., "Data Compression and Data Compaction", NTIS Technical Report AD 723525, 1972

69. Ruth,S.R.,Kreutzer,P.J., "Data Compression for Large Business Files", Datamation, Vol 18(Sept. 1972), pp 62-66

70. Salton,G.A., "Computer Evaluation of Indexing and Text Processing", Journal of the ACM, Vol 15(1968), pp 8-36

71. Salton,G.A., "Automatic Information Organization and Retrieval", McGraw Hill, New York 1968

72. Salton,G.A.,Yu,C.T., "On the Construction of Optimal Vocabularies for Information Retrieval", Paper presented at the SIGIR-SIGPLAN Interface Meeting Nov 1973, Gaithersburg,Md.

73. Schieber,W.D.,Thomas,W.G., "An Algorithm for Compaction of Alphanumeric Data", Journal of Library Automation, Vol 4(1971), pp 198-206

74. Schipma,P.B., "Term Fragment Analysis for Inversion of Large Files", Paper presented at the Association of Scientific Information Center Meeting Washington DC Feb. 1971

75. Schuegraf,E.J.,Heaps,H.S., "Selection of Equifrequent Word Fragments for Information Retrieval", Information Storage and Retrieval, Vol 9(1973), pp 697-711

76. Schwartz, E.S., "A Dictionary for Minimum Redundancy Coding", Journal of the ACM, Vol 10 (1963), pp 413-439

77. Schwartz, E.S., Kallick, B., "Generating a Canonical Prefix Encoding", Communications of the ACM, Vol 7 (1964), pp 166-167

78. Schwartz, E., "An optimum Encoding with minimum longest code and total number of digits", Information and Control, Vol 7 (1964), pp 37-44

79. Schwartz, E.S., Kleiboemer, A.J., "A language element for compression coding", Information and Control, Vol 10 (1967), pp 315-333

80. Scidmore, A.K., Weinberg, B.L. "Storage and Search Properties of a tree organized memory system.", Communications of the ACM, Vol 6 (1963), pp 28-31

81. Shannon, C.E., "A mathematical Theory of Communication", Bell System Technical Journal, Vol 27 (1948), pp 379-429, 623-656

82. Shannon, C.E., "Prediction and Entropy of printed English", Bell System Technical Journal, Vol 30 (1951), pp 50-64

83. Siler, K.F., "A Stochastic Model for the Evaluation of large scale data retrieval systems: An Analysis of Database Inversion and Key Truncation", PH.D. Thesis UCLA, 1971

84. Snyderman, M., Hunt, B. "The Myriad Virtues of Text Compaction", Datamation, Vol 16 (Dec. 1970), pp 36-40

85. Stockmal, F., "Generation of Partitions in Part Count Form (Algorithm 95)", Communications of the ACM, Vol 6 (1962) p 344

86. Svenonious, E., "An Experiment in Index Term Frequency", Journal of ASIS, Vol 23 (1972), pp 109-121

87. Thiel, L.H., Heaps, H.S., "Program Design for Retrospective Searches on Large Databases", Information Storage and Retrieval, Vol 8 (1972), pp 1-20

88. Treleaven, R., "Abbreviation of English Words to Standard Length for Computer Processing", M.Sc. Thesis, University of Alberta Edmonton, 1970.

89. Walker, V.R., "Compression of Names by X-grams", *Proceedings of ASIS*, Vol 6 (1969), pp 129-135

90. Wagner, R.A., "Common Phrases and Minimum Text Storage", *Communications of the ACM*, Vol 16 (1973), pp 148-153

91. Wagner, R.A., "An Algorithm for Extracting Phrases in a space optimal Fashion", *Communications of the ACM*, Vol 16 (1973), pp 183-185

92. Wells, M., "File Compression using variable length encodings", *Computer Journal*, Vol 15 (1972), pp 308-313

93. White, H.E., "Printed English Compression by Dictionary Encoding", *Proceedings IEEE*, Vol 55 (1967), pp 390-396

94. Yu, C.T., "Theory of Indexing and Classification", *Ph.D. Thesis Cornell University*, TR 73-181

95. Zipf, G.K., "*Human Behaviour and the Principle of least Effort*", Addison Wesley, Cambridge Mass. (1949)

96. Zunde, P., Slamecka, V., "Distribution of Indexing Terms for Maximum Efficiency of Information Transmission", *American Documentation*, Vol 18 (1967), pp 104-108

APPENDIX I

TEXT FRAGMENT DICTIONARY

SAMPLE 1 t= 10

```
 1[illegible] _[illegible]        [illegible] _ADD     [illegible] _AM
 11 _AMERICAN        1[illegible] _ANA          1[illegible] _AND_
 2[illegible] _ANA[illegible]A          1[illegible] _ANDAC        1[illegible] _AND_E
 11 _ANGLUES         [illegible] _ANGLIC       13 _AND_THE_
 11 _ANT             1[illegible] _APP          1[illegible] _ART
 12 _AT              1[illegible] _AUT          1[illegible] _[illegible]
 1[illegible] _BRIT             15 _CAR          1[illegible] _CHA
 10 _CHI             11 _CIVIL        1[illegible] [illegible]
 24 _COMP            12 _CON          13 [illegible]
 11 _CONT            10 _CRITICISM    10 _DAVI
 12 _DIS             11 _ECO          11 _EDWARD
 10 _EM              10 _ENG          10 _ETC
 30 _FOR_            14 _FOR_         12 _FRAN
 11 _FRE             10 _FROM_        17 _GOVE
 13 _GRA             13 _H.           10 _HAR
 16 _HISTORY.        10 _HU           13 _IN_A
 11 _IN_E            20 _IN_THE       11 _INDUSTR
 12 _INST            14 _INTER        10 _JOHN
 14 _LANGUA          15 _LAW_         23 _LAW
 32 _LECTURES.       12 _LITERATUR    10 _MAR
 12 _MAT             13 _MODERN_      12 _NAT
 30 _NEW_            10 _OF           14 _OF_E
 13 _OF_F            11 _OF_M         11 _OF_MA
 16 _OF_P            13 _OF_S         24 _OF_THE
 10 _ON              10 _OP           10 _PAUL_
 15 _PER             15 _PHIL         10 _PL
 14 _PLAN            11 _POW          13 _POLITIC
 13 _PRE             10 _PUBLIC_      11 _READ
 12 _RELATIONS       12 _REP          10 _RESEARCH
 10 _SAM             12 _SCHOOL       11 _SEC
 12 _SERVICE         12 _SOCI         11 _SOCIAL_
 14 _SOUTH           12 _STAT         12 _STATES
 11 _STORY           12 _STUD         43 _STUDY
 11 _TEACH           12 _THE_C        10 _THE_CO
 12 _THE_D           12 _THE_M        10 _THE_N
 11 _THE_W           26 _TRA          16 _U.S._
 19 _U.S.            10 _WALTER       12 _WAR
 10 _WILLIAM         12 _WITH_        14 _WORK
 35 '                41 )             58 :
 13 :A S             12 :AN           27 ^
 31 (                11 (S           572 ,
 10 ,COMMITTEE       12 .H            11 ,S
 13 &                93 -              1 /
  7 ;                18 ,_AL          14 ,_CHARLES_
 13 ,_DA             11 ,_DO          32 ,_ESSAYS,
 12 ,_GE             12 ,_GEORGE_     10 ,_HE
 10 ,_JAMES_         21 ,_JOHN_       15 ,_JOSEPH_
 11 ,_MAR            11 ,_[illegible]         10 ,_PA
 1[illegible] ,_RICHARD         12 ,_RO          19 ,_ROBERT_
 14 ,_SA             11 ,_ST          1[illegible] ,_WA
 17 ,_WILLIAM_       14 ,_196         15 ,_1968.
  6 A_               11 A_AND_        10 A_STUDY
 10 ACCO             11 ACHE          21 ACHIN
 14 ACK              31 ACTI          10 ADA
```

[illegible]	[illegible]	10	AEL	[illegible]	[illegible]
10	[illegible]	[illegible]	AL_C	[illegible]	[illegible]
10	AEED	11	ALE_	[illegible]	[illegible]
10	ALLRE	11	ALI	[illegible]	AUE
[illegible]	AGOE	12	ALTS	[illegible]	AVER
10	AMERICAN	17	[illegible]	13	AN_
[illegible]	ANALYSIS	14	ANCE_	11	ANCI
12	AND	13	AND_O	12	ANDER
12	ANE	16	ANGE	13	ANI
10	ANIC	11	ARCH	30	ARD_
12	ARIE	12	ARING	13	ARK
10	ARKE	11	ARM	16	ARR
11	ARRI	10	ARS	12	ARTH
10	ARY_	10	ASHI	11	ASI
10	ASP	14	ATE_	12	ATE
10	ATED_	20	ATER	24	ATION
11	ATIONAL	15	ATIONS	15	[illegible]
12	ATU	10	AUR	10	AUT
15	AVI	248	B_	11	B.
10	BAL	12	BERT_	11	BIL
11	BIN	11	BLI	10	BOOK
13	BRA	10	BRE	12	BRO
12	BU	10	BUS	13	BY
12	C	10	CAN_P	11	CAR
10	CAS	10	CASE	10	CATION
11	CE OF	1[illegible]	CE.	12	CENT
12	CES	10	CESS	13	CH_
10	CHANG	10	CHAR	10	CHILD
11	CHN	12	CHR	11	CHRISTIAN_
10	CIN	11	CKE	10	CLAS
16	COLLE	10	COMME	10	COMMI
12	COMMITTEE	10	CONGRESS,	10	COU
14	COUNT	11	CTION,	11	CTIV
14	CTOR	11	CUL	10	CULT
10	CUR	17	D	12	D_AN
10	DEF	12	D_IN	11	D_HE
10	D_PO	12	D_TO	24	D,
12	DAN	12	DAT	12	DD
11	DEA	11	DEL	14	DENT
12	DER_	14	DERS	11	DES
11	DEVELOPMEN	12	DG	10	[illegible]
13	DI	10	DING	11	DL
12	DN	11	DOC	10	DOR
32	DRESS[illegible],_E	10	DS	10	DS,
11	DW	20	DY	2	E_
10	E_AND	12	E_COM	12	E_CON
13	E_DE	13	E_IN	11	E_IN
10	E_L	13	E_MA	12	E_PO
13	E_PRO	10	E_R	10	E_THE
10	E_U.S.	10	E_.	13	EAC
11	EADER	11	EBRA	15	ECONOMIC
11	ECTION	11	ECTS	13	ED.
11	ED_STATES_	1[illegible]	EDE	1[illegible]	ED
11	EDUCATION_	[illegible]	EDUCATION-	10	EE_ON_

12	[illegible]	16	ECI	[illegible]	[illegible]
12	[illegible]	[illegible]	ELOGY	[illegible]	[illegible]
10	EMA	12	EME	13	[illegible]
[illegible]	EMS	14	ENS	10	[illegible]
10	ENCE	16	ENGLISH_	11	[illegible]_A
12	ENTS_	11	EPA	10	[illegible]
14	ER	11	ER,_R	13	ERAL
10	ERIA	11	ERIC	12	ERIN
17	ERN_	12	ERR	13	ERS,_
14	ERS,	10	ERSI	13	ERV
10	[illegible]	19	ES_AND_	11	ES_
10	ES_IN	20	ES_OF_	13	ESI
10	ES,_ETC,	11	ESE_	13	ESE
10	ESI	16	ESSI	14	ESTI
11	ETE	11	ETH_	10	ETTE
12	EUR	10	EVA	11	EVEN
16	EXA	17	EY,	19	F
10	F_OO	14	F,	10	FEDERAL
14	FERN	30	FF	11	FIC
12	FIE	12	FIR	10	FIS
19	FL	17	FO	13	FORE
13	FORM	12	FU	15	G
10	G_S	30	[illegible]	14	G,
11	GE_R	13	GEN	10	GENE
11	GES_	11	GHTS	12	GIN
16	GL	11	GOR	12	GRAMM
10	GRE	17	GRESS	26	GU
13	GUI	10	GUIDE_TO_	10	GY
38	H	10	H_[illegible]	28	H,
12	H,	15	HAN	12	HANI
10	HARD	22	HE_P	12	HEA
10	HEAR	13	HEI	11	HES
11	HIN	10	HI[illegible]	10	HIS
11	HISTOR	16	HISTORY_OF	10	HISTORY_A
10	HOR	10	HORS	12	HOU
16	HOW	16	HU	12	HUR
19	HY	2	I	13	IA
13	IAL_	17	IAL_S	17	IAN_
15	ICAL	14	ICAL_A	10	ICES
13	ICH	13	ICO	10	ICS_
20	ICS,	12	ICTION	11	IDA
17	IDE	16	IEL	16	IENCE
10	IER	11	IES_	10	IES,
20	IGH	12	IGN	10	ILE_
21	ILI	40	ILL	11	IME
11	IN,	10	INA_	[illegible]	INE_
10	INEE	26	INES	21	ING_
14	ING_A	12	ING_C	10	ING_TH
10	INS_	10	INTE	12	INTER
12	IOLOGY	10	ION	11	ION_A
11	ION_T	10	IONAL_C	10	IOUS_
10	IP	12	IRS	13	IS,_
10	ISC	11	ISCO	21	ISH
10	ISH_LANGUA	10	ISI	11	ISH_

```
10 ISON          13 ISTI          10 ISTO
10 ITA           12 ITAL          11 ITS_
16 ITY_          25 IVE_          10 IVERS
72 J             15 J,            35 JU
84 K             14 K,            21 K,
11 KEN           10 KER,          10 KING
11 KINS          10 KL            22 L
13 L_AND_        11 L_SE          10 L,
10 LABOR         11 LAC           21 LAND
11 LAR           17 LATI          10 LAU
11 LAWS_         11 LB            15 LEA
10 LECTION       10 LEGISLATIO    13 LEN
12 LET           15 LEY           17 LF
11 LIFE          10 LIP           12 LISM
11 LK            14 LL,_          13 LLA
12 LLI           10 LLO           13 LOC
14 LOG           11 LOR            5 M
16 M,            13 MAC           18 MAN_
17 MAN,          11 MANAGEMENT    11 MAR
10 MAS           10 MAT           10 MATHE
10 MATIC         14 MB            11 MEDI
11 MENT,         14 MENTA         11 MER
10 MERC          10 MERICA        10 MES,_
10 METR          10 MICS          23 MIL
10 MINA          11 MINATIONS     15 MIS
16 MISSION       13 MIT           12 MM
10 MMAR          10 MOD           14 MOR
11 MPA           12 MPL           26 MY
 0 N             11 N_AN          12 N_AND_
16 N_CO          10 N_INT         13 N_LA
10 N_LI          10 N_OF_THE      14 N_PO
11 N_RE          14 N_THE_        14 N,_A
10 N,_P          11 N,_ROBERT_    12 NAL
11 NAMES         18 NANC          12 NARY_
19 NATIONAL_     11 NCE,          15 NCES
12 NCI           10 ND_G          11 NDS
10 NE,_          14 NER,_         11 NESS
13 NET           10 NEW           11 NF
12 NGIN          10 NGS_          13 NIN
12 NING          12 NIST          10 NIT
10 NIZATION      11 NMENT         10 NNING
13 NOM           11 NOR           10 NSE
14 NSON,_        12 NSTITUT       11 NT_OF_
12 NTIN          11 NTO           11 NTRO
10 NTS           11 NVE           16 O
20 O,            13 OBI           10 OCA
10 OCK           12 ODE           10 ODO
10 OF_AMERICA    19 OGRAPHY       11 OH
12 OK            12 OLD_          15 OLE
17 OLI           10 OLO           10 OLOGICAL
20 OLOGY_        14 OLOGY,        14 OLS
17 OM            11 OMA           14 OME
10 OMET          15 OMI           12 ON_G
10 ON_IN         14 ON_P          10 ON;
```

```
12 ONA              11 ONALD            11 ONE
10 ONS_OF_          10 ONTI             10 OOD
13 OPERATION        13 OPH              10 OR_THE_
11 ORE_             10 ORG              11 ORGANI
11 ORN              10 ORS              10 ORY
12 OSE              12 OTH              19 OU
12 OUGH             13 OUN              12 OUT
10 OVE              12 OVERNMENT_       16 OW
15 OWE              11 OWN              10 OY
 2 P                11 PAI              12 PARA
14 PEA              12 PEN              11 PERS
10 PERSONAL         12 PHE              12 PHI
10 PHY.             10 PL               11 PLE
13 PO               10 POET             11 POETRY_
11 POLIC            20 PORT             12 POS
17 PP               10 PRES             12 PROCE
11 PROG             13 PSYCHO           21 PU
 8 Q                15 QUE               3 R
11 R_AND_           10 R_RE             10 R;
10 R,_E             10 RA_              10 RAC
13 RACT             10 RAD              11 RADE
13 RAI              13 RAL              13 RANC
11 RAR              11 RATE             12 RATIONS
12 RD,_             14 RE.              12 RE,_
12 REAT             11 RED_             10 RED
14 REG              13 RELIGIO          14 RENCE
13 RET              10 REV              14 RGE
11 RIAL_            10 RICA_            11 RIES_
14 RIES,            12 RING_            12 RIST
13 RITI             12 RNA              10 RNS
12 RODUCT           11 RON              10 RONIC
10 RRI              12 RSO              11 RST
11 RTE              10 RTH_             11 RTI
12 RTS              15 RY_OF             1 S
11 S_AND_T          12 S_CO             14 S_FOR_
10 S_L              10 S_O              13 S_OF_A
12 S_OF_THE         17 S_ON             10 S_R
15 S_U.S.           10 S_W              14 S;A
10 S.C              15 S,_A             15 S,_C
10 S,_ET            10 S,_P             15 S,_S
10 SCA              11 SCIEN            13 SEA
13 SEL              10 SES              14 SES,
10 SHING            10 SHO              10 SIA
10 SIG              14 SING             10 SION.
12 SIONS            17 SLA              10 SOC
11 SOCIAL           12 SON_             24 SON,_
12 SOUR             12 SPA              14 SPECT
14 SS,_             13 SSA              11 SSE
10 SSES             11 SSION_           11 ST,_
13 STA              15 STAN             10 STATE_
22 STER             11 STIN             10 STRA
11 STRAT            10 STRUCT           12 STRY_
11 STU              14 SUR               6 T
13 T_AND_           11 T_IN             13 TAL
```

```
 1Ø TALY             1Ø TARY_            14 TATION
 11 TEC              1Ø TEL              11 TEM
 13 TERI             1Ø TERN.            1Ø TERNATI
 11 TEXT,            1Ø TH_A             16 TH_CENTURY
 1Ø TH_CONGRE        11 TH,              1Ø THE
 15 THE_A            11 THE_F            1Ø THE_F
 17 THE_L            12 THE_M            12 THE_R
 17 THE_S            12 THE_ST           11 THE_UNITED
 12 THEOR            12 THER             1Ø THERN
 11 THOD             1Ø THOMAS           1Ø THR
 1Ø TIA              16 TIC_             17 TICAL_
 16 TICS_            12 TIES             16 TIES
 15 TING_            16 TION_AND_        11 TION_OF_
 1Ø TIONAR           16 TIONS,_          1Ø TO_T
 11 TOL              12 TON_             13 TON,_
 1Ø TORY             1Ø TRAIN            11 TRIC
 13 TRON             13 TS_AND_          12 TS_OF_
 1Ø TS,_             1Ø TTE              18 TURE_
 16 TURE            1Ø1 U                22 U.S.
 18 UAGE_            11 UAL              24 UB
 13 UD               1Ø UDEN             1Ø UDIE
 11 UEL              12 UES              23 UG
 16 ULA              13 UNC              14 UP
 14 URA              14 URC              1Ø URE
 12 USTR             1Ø UTHER            33 V
 12 VAL              1Ø VAN              11 VEL
 12 VENT             11 VERN             11 VIE
 15 VIL              13 VIS              14 VO
 92 W                16 W.               1Ø W,
 11 WAR              11 WAS              11 WER
 1Ø WES              14 WH               15 WIS
 14 WORLD_           61 X                1Ø XIC
 62 Y                11 Y_AND_           11 Y_AND_C
 12 Y_AND_T          12 Y_IN_            17 Y_OF_THE_
 21 Y;               62 Y.               12 YL
 14 YM               12 YO               12 YOU
 1Ø YS               13 YST              5Ø Z
 44 !                18 Ø                11 Ø-
  6 1                24 2                12 2Ø
 23 3                27 4                33 5
 16 6                1Ø 68               24 7
 21 8                44 9                12 96
```

APPENDIX II

WORD FRAGMENT DICTIONARY

SAMPLE 2   t=672

```
  704             2556 'S             270 A
 1254 ACE          765 ACH            782 ACHI
  698 ACK          732 ADE           15[illegible] ADE
 2138 AGE          867 AGRIC          784 AIL
 1754 AIN          956 AIR            782 AKE
  672 ALA         1302 ALE           1824 ALI
  714 ALIS        2283 ALL           1894 ASS
 1367 AME          763 AMERICA       1674 AMERICAN
 1411 AMI          674 ANALYS        2380 ANCE
18193 AND          773 ANGE          2565 ANI
 1039 ANN          876 ANS           2792 ANT
  718 APH         1186 APP           1191 ARA
  770 ARCH        1488 ARD           1623 ARE
 2199 ARI         3120 ART           3396 AS
 1141 ASE         1818 ASS           1959 ATE
  683 ATER         809 ATHE          1050 ATIO
  787 ATIONAL     2005 ATIONS         920 ATIVE
 1258 ATO         1055 AUT           1216 AV
  709 AVE          804 AYS           9193 B
 1248 BER         1035 BIL           1530 BLE
 2508 BO          1523 BOOK           827 BRA
  693 BS          1661 BU             746 BUS
  816 BY           339 C             1345 CA
 1102 CAL          696 CAP           1168 CAR
  684 CAS         1129 CAT           1078 CENT
 2160 CES          768 CESS          1265 CHA
 1008 CHAN        1215 CHE            925 CHEMI
 1879 CHI          826 CHILDREN       672 CHRIST
  675 CHUR         826 CIAL           688 CIVI
  684 CK          2133 CO            1201 COL
 1090 COLLE       1128 COMM           938 COMMUNI
 1760 COMP        2925 CON           1087 CONS
  916 CONTR        867 COO           1167 COR
 1087 COUNT        685 CRIM          1123 CTI
 1000 CTION        730 CTIONS         978 CTOR
 1044 CTURE       1145 CUL           7000 D
  795 DEN          936 DENT          1794 DER
 1344 DES         1413 DIA            851 DICA
 1635 DING        1767 DIS           2781 DO
 3254 DS           688 DU[illegible]  728 DUCTION
  889 DY           221 [illegible]   2561 EAR
 2041 EAS         1270 ECO           1388 ECONOMIC
  708 ECTI         924 ECTION         673 EDE
 1374 EDI         2280 EDUCATIO       856 EED
  856 EGE          921 ELE           1181 ELECT
  676 ELECTRON     762 ELI           1299 ELL
  942 EMA         1533 EMENT          987 EMP
 1290 ENCE        1178 END            936 ENE
 1627 ENGLISH      735 ENI           1247 ENS
  792 ENT          788 ENTA          1169 ENTI
  710 ENTU        1162 ERAL           958 ERAT
  675 ERI          673 ERIA           774 ERING
  784 ERNA        3045 ERS           1028 ESE
 1034 ESI          913 ESSI          3266 EST
 1183 ETA            1180 ETE            1440 ETI
  975 ETR             686 ETT            1017 EVE
  903 EVELOPME        789 EXP           37126 F
  672 FAC            1106 FF              723 FFE
 1662 FO             4445 FOR             970 FORE
 1018 FORM           1067 FROM           1314 FU
10553 G               855 GEN             684 GENE
  922 GEO             734 GEOLOGY        1164 GES
  661 GH              707 GIN             786 GION
  725 GOVERNME       1755 GRA            1371 GU
  810 GUIDE          2408 H               803 HAND
 1193 HAR             753 HEM            1606 HER
  760 HES             629 HIGH            993 HIN
  961 HING           1556 HISTORY        1177 HO
  716 HOL             741 HOLOGY         1127 HOR
 1319 HU             1048 HY               80 I
 1366 IAL            1969 IAN             766 IANS
 3170 ICAL            872 ICATION        1594 ICI
 1536 IDE            1916 IES            1146 IGHT
 1020 ILE             978 ILI            1222 ILIT
 1630 ILL            1482 INA            1200 INC
 1378 IND             714 INDIAN          674 INDUS
  861 INDUSTRI       3367 INE            1364 INES
 1430 ING            1334 INGS            720 INING
 1208 INS             711 INST            852 INTER
  674 INTERN          746 IOLOGY          700 IONAL
 1424 IONS           1011 IRE             891 ISE
 1832 ISH            1929 ISM            1669 IST
 1552 ISTR           1371 ITA             758 ITAL
 1396 ITE            1326 ITIES           878 ITION
 1336 ITY             695 IUM             883 IVER
  672 IVIL            966 IZATION        3201 J
 1061 JU             3962 K              1160 KING
 1478 KS              343 L               788 LABOR
 1995 LAND           1147 LANGUAGE        949 LAR
 1414 LAS             858 LATION          725 LATIONS
 1915 LAW            1053 LEA             782 LECT
 2521 LES             675 LET             706 LF
 1051 LIA             728 LIBRAR          804 LIC
 1215 LIFE           1534 LIN             775 LING
  730 LIS            1199 LITERATU        824 LITY
  866 LLE            3533 LO              716 LOG
 1098 LOR             803 LOS             704 LOW
  353 M              1135 MAL            3555 MAN
  948 MANA           1913 MAR            1558 MAT
  701 MATI            785 MATIC           733 MEDIC
 1227 MEN            2507 MENT            840 MENTAL
 1318 MENTS          1304 MER            1900 MET
  815 MIC             798 MICS           1418 MIN
 1012 MINA            900 MINI           1446 MIS
 3346 MO              833 MODERN         1390 MON
  873 MOR             950 MPL            1065 MU
  735 MUSIC           658 N               851 NAGEMENT
 1538 NAL             690 NANC            771 NATION
```

1[illegible]	NATIONAL	1478	NC	[illegible]75	NCE
730	NEER	764	NEGRO	968	NER
694	NERA	987	NESS	808	NET
1332	NG	749	NGIN	1542	NIC
1311	NING	671	NIS	868	NIST
706	NITY	788	NNE	720	NOM
701	NORTH	786	NSI	1277	NTE
756	NTING	1068	NTRO	158	O
1839	OC	721	OCA	739	OCK
2574	OD	1222	OG	1249	OGRAPHY
879	OLD	860	OLE	842	OLIC
723	OLOGI	2118	OLOGY	1086	OMA
2277	OME	4218	ON	1328	ONA
1307	ONE	826	ONG	790	ONIC
721	ONO	821	ONS	816	OO
1155	OOD	760	OOK	3143	OP
866	OPER	1086	ORD	1128	ORE
1126	ORI	1365	ORS	814	ORT
1567	OS	748	OSI	850	OTH
752	OTO	995	OUND	1824	OUR
1510	OUS	1120	OUT	1380	OVE
1411	OW	766	OWER	704	OWN
255	P	672	PAC	1755	PAR
786	PEN	1745	PER	1076	PERS
714	PHI	704	PHIL	697	PHO
693	PHY	958	PHYSIC	710	PIN
918	PLA	1556	PLAN	1172	PLE
733	PLO	2753	PO	761	POETRY
705	POL	834	POLI	785	POLITIC
1137	PORT	982	POS	1170	PP
930	PRE	892	PRES	829	PRI
722	PRIN	3337	PRO	711	PROB
1050	PROCE	819	PROGRAM	1253	PS
775	PSYCHO	1128	PUBLIC	40	Q
727	QU	911	QUA	1028	QUE
243	R	782	RAC	1178	RACT
869	RADI	752	RAIN	1810	RAN
934	RAP	1782	RATION	1438	REA
3312	REA	937	REAT	937	REAT
1493	REC	931	RED	922	RELAT
718	RELI	983	REN	817	RENC
893	REPORT	880	RESEARCH	680	RESO
887	RESS	739	REST	1030	RET
974	RIA	1022	RIC	877	RICULTUR
707	RIE	1417	RIES	837	RING
1193	RITI	1492	RMA	1049	RODU
769	ROL	756	ROM	1383	RON
907	ROP	801	ROPE	742	ROS
1170	ROU	701	RTH	2940	S
758	SAL	868	SCHOOL	1133	SCIENCE
689	SCO	911	SEL	791	SERVICE
1706	SES	893	SH	724	SHE
742	SHIP	695	SHO	1318	SING
1187	SION	1001	SIS	826	SK

```
  791 SLA          682 SLAVE        88[illegible] SM
 2451 SO           767 SOCI        1160 SOCIAL
 1684 SON          672 SOURCE       741 SOUTH
 1186 SP           732 SPA          864 SPE
  729 SPECT        827 SPO          841 SSI
  981 SSION       1052 STAN        1074 STATE
 1111 STATES       856 STE         1409 STER
 1422 STIC        1422 STIC         828 STO
  934 STOR         682 STORIES      826 STRA
[illegible] RATI   791 STRUCT       719 STUDY
  729 [illegible]  850 SUP          850 SUP
  732 [illegible] 1129 SY           932 SYSTEM
  838 [illegible] 1597 TAL          700 TAN
[illegible]       1194 TATION       782 TECHN
[illegible]       1066 TEN          984 TERI
[illegible]       1447 TES        27848 THE
[illegible] TH[illegible]RY 1607 THER  975 THI
[illegible] [illegible]O  1302 TICAL  727 TIN
 96[illegible] [illegible]ND 1102 TIO  678 TIST
[illegible] [illegible]E 1078 TOM  1069 TON
[illegible] TORI  2201 TRA          867 TRADE
[illegible] TRANS  788 TRIAL       1048 TRIC
[illegible] TROL   730 TTER        1001 TURAL
 2850 TURE          81 U"          1014 UB
 1114 UC          1935 UD          1326 UG
  747 UL          1194 ULA          852 ULT
 2984 UM           690 UNDER       1035 UNITED
  745 URCH         795 URES        1085 US
 1159 USE          839 USIN        1120 USTR
  675 UTH         1160 UTION       7848 V
  681 VEL          938 VERS         700 VERY
 1154 VES        10269 W           1218 WATER
 1008 WH           798 WIN          879 WITH
  984 WORK        1108 WORLD        816 WS
 4259 X          23702 Y            903 YOU
  689 YSIS        2290 Z           2023 0
  721 1           1361 18          2127 19
[illegible]44 196 1201 2           1350 3
 1531 4           1939 5           1570 6
 1939 7           1277 8           1277 9
```