[From: Internat.conf. *Methodology & Techniques of Machine Translation*, British Computer Society, Cranfield, 13-15 February 1984]

SEARCHING SINGLE-WORD AND MULTI-WORD DICTIONARIES

F.J. Smith, K. Devine and P. Craig.

Department of Computer Science

Queen's University of Belfast

N. Ireland.

Abstract.

We have developed fast methods for retrieving records from large single-word or phrase dictionaries. We have also computed the Zipfian law for word pairs and for 3-word phrases and have deduced that the number of multi-word phrases which occur frequently in text is relatively small, of the same order of magnitude as the number of individual words in a text. They are thus amenable to computer processing and may help machine translation and knowledge based systems.

Introduction.

In information retrieval (IR), a corpus of text is indexed for fast access so that all segments of text containing a given word can be quickly retrieved. Research on IR has been carried out at Queen's University since the mid-1960's with emphasis on systems design (Higgins and Smith, 1969; Devine and Smith, 1984). For these systems we have developed fast methods of retrieving individual word records from a large dictionary of such records. Similar tasks must arise in M.T.

We have also developed methods of storing dictionaries of phrases. We have felt that the use of single words to represent knowledge was inhibiting our research, as it was evident that the human brain assimilates much knowledge from text in phrases made up of groups of words. We suggest that our knowledge of a language is made up of a large number of such groups of words, each representing one unit of information. Although this unit may be subdivided into smaller word units it is not so subdivided consciously when communicating information. (Indeed a child may be unaware of the subdivision without affecting its understanding).

We believed it possible by the techniques we will describe, to store large numbers of such phrases; but first we needed to know their frequency in text corpora and some automatic method to help generate them. This paper describes our work on word phrases; but first we present the storage techniques used for storing single words.

1. Dictionary Files on Disk - Methods of Access.

1.1 Alphabetically-ordered files.

There are a number of ways in which the dictionary file may be organized to permit fast retrieval of the record for a specific word. In the simplest method, the records may be sorted into alphabetic order, and stored in this order on a random-access file, with a certain number, b, of records per block. A block is a file area of fixed size, typically 512 characters. All file accesses cause one or more whole blocks to be read or written, and it is the number of such file accesses required to find the desired record which determines the speed of response.

If there are N words (and hence N records) in the file, the number of blocks will be $B = N/b$. For a linear search of the file, on average B/2 read accesses will be required to find a desired record. For a binary search of an alphabetically-ordered file, the average number of accesses $\bar{a}$ is

$$\bar{a} = \text{Log}_2\ (B/2)$$

(Martin, 1975). With N = 10,000 and b = 20, this is 8 accesses.

The most popular method of accessing alphabetically-sorted files is the indexed-sequential method, in which a number of blocks are reserved to act as a directory or index to the rest of the file, by recording for example the identity of the first word in each block (Martin, 1975). The number of accesses is now limited by the number of index blocks which must be read to find the record. For a large file a further reduction can be achieved by making an index to the index, and so on until the top-level index fits in a single block (this is a multi-level index sequential file).

One of the problems with an alphabetically-ordered file concerns updating. If a new word is added to the file, it must be inserted in its proper position. Material which follows it must be pushed down, and the indexes must be revised. This problem can be largely overcome by implementing the order of blocks (both index and data blocks) through pointers rather than by physical positioning. Each block must still contain a set of alphabetically-adjacent records, whose internal order may also be established through pointers. When such a block fills, it can be split in two with limited repercussions on the organization. This is the B-tree principle (Ore, 1979), and represents the highest development of alphabetically-ordered files.

1.2 Hash-ordered files.

But none of the alphabetically ordered structures are as fast as hash ordered files, particularly when the files are large. In this structure the records are assigned to blocks according to a division hashing algorithm (Maurer, 1968). We must estimate in advance how many blocks are required (say, B, which should be an odd number). We divide a numerical representation of the key (word) by B, and use the remainder as the block number. There are many ways to derive a numerical value from the key, and one must be chosen which will distribute the records uniformly over the available blocks. One possibility is to use the ordinal values of the characters c_i in the ASCII sequence (or some other sequence), e.g.

$$K = \text{ord}\ (c_n) + 256\ \text{ord}\ (c_{n-1}) + 256^2\ \text{ord}\ (c_{n-2}) + \ldots \qquad (1.1)$$

Then the block number is

$$h_0(K) = K \bmod B \qquad (1.2)$$

In a growing file a stage will soon be reached where a new record is assigned to a block which is already full - an overflow situation.

1.2.1 Overflow.

Overflow can be managed by extending the file and using the new blocks at the end to hold the overflowing records. However if the overflow area is allowed to grow in this way performance will suffer. The overflow area may also be too small. So the file may eventually have to be reorganized even though the whole file may have a low packing density with many blocks only partially filled.

Other overflow methods consist of algorithms which direct an overflow record to a different hash block $h_1(K)$, or if this too is full to $h_2(K)$ and so on. The file remains the same size until it fills completely.

There are a number of these methods, the simplest being the linear probing method (Peterson, 1957). In this a block overflows into the next adjacent block:

$$h_i(K) = (h_0(K) + i) \bmod B \qquad (1.3)$$

This can produce sequences of adjacent full blocks due to a cascade effect, which slow down performance. It is therefore not recommended. This cascade effect can be overcome by a number of techniques including quadratic probing (Maurer, 1968) where

$$h_i(K) = (h_0(K) + ai + bi^2) \bmod B$$

and a and b are constants. A simple variant is the linear quotient method (Bell and Kaman, 1970). In this quotient method we put $b = 0$ and $a = Q$ where Q is the quotient obtained after dividing $h_0(K)$ by B;

$$h_i(K) = (h_0(K) + i\,Q) \bmod B \qquad (1.4)$$

If $Q = 0$, we use the value $Q = 1$ instead. This method is particularly simple and we have found it effective in practice. However, B must be prime to ensure that all blocks are probed by the sequence $h_i(K)$.

3. Word Pairs.

At first the storage of a dictionary of word pairs seems a formidable task because of the large number of possible pairs which can occur. If we assume that we will build a system to cope with a dictionary of 30,000 word types, then there are 900 million possible word pairs. Fortunately the number of pairs which need to be considered is very much smaller because many word pairs cannot occur in correct English. For example, the pair "the of" cannot occur. Our task is also made easier because we are primarily interested in pairs which form a syntactic or semantic unit or part of such a unit and therefore occur frequently. Most word pairs in correct English do not have this property and occur very infrequently and are of less interest.

We begin by looking at the frequencies of pairs.

3.1 Frequencies.

To investigate the frequencies of word pairs (and other multi-word combinations) we examined a file of legal text of 16,050 tokens (i.e. word occurrences) containing 2,615 different word types. We first computed the frequencies, f, of each word and ranked the words in order with r = 1 for the most frequent, r = 2 for the 2nd most frequent, and so on. The graph of Log f, against Log r in Figure 1 shows it follows a straight line with a slope close to -1, the well known Zipfian distribution (Zipf, 1949; Fedorowicz, 1982):

$$f_1 = \frac{K_1}{r} \qquad (3.1)$$

where K_1 is a constant which is proportional to the size of the text. For our text it is approximately 1300, which is close to the frequency of the most common word, "THE".

We did the same for all of the word pairs in the text and computed their frequencies f_2; the plot of Log f_2 against Log r is also shown in Figure 1. The curve can be approximated by a straight line with a slope just less than $\frac{2}{3}$ and with a maximum frequency of about 250 for the pair "OF THE", much less than the maximum frequency of the most common word, THE, which is about 1300. For word pairs in our sample text we found

$$f_2 = \frac{200}{r^{0.65}}$$

Our sample is large enough for us to expect that this law will hold for all English texts; so

$$f_2 = \frac{K_2}{r^{\beta}} \qquad (3.2)$$

where $\beta \approx 0.65$ will represent Zipf's law for word pairs and $K_2 = K_1/6.5$. We are carrying out more tests on larger bodies of text to calculate the coefficients β and K_2/K_1 more accurately and to check that they are invariants for all text.

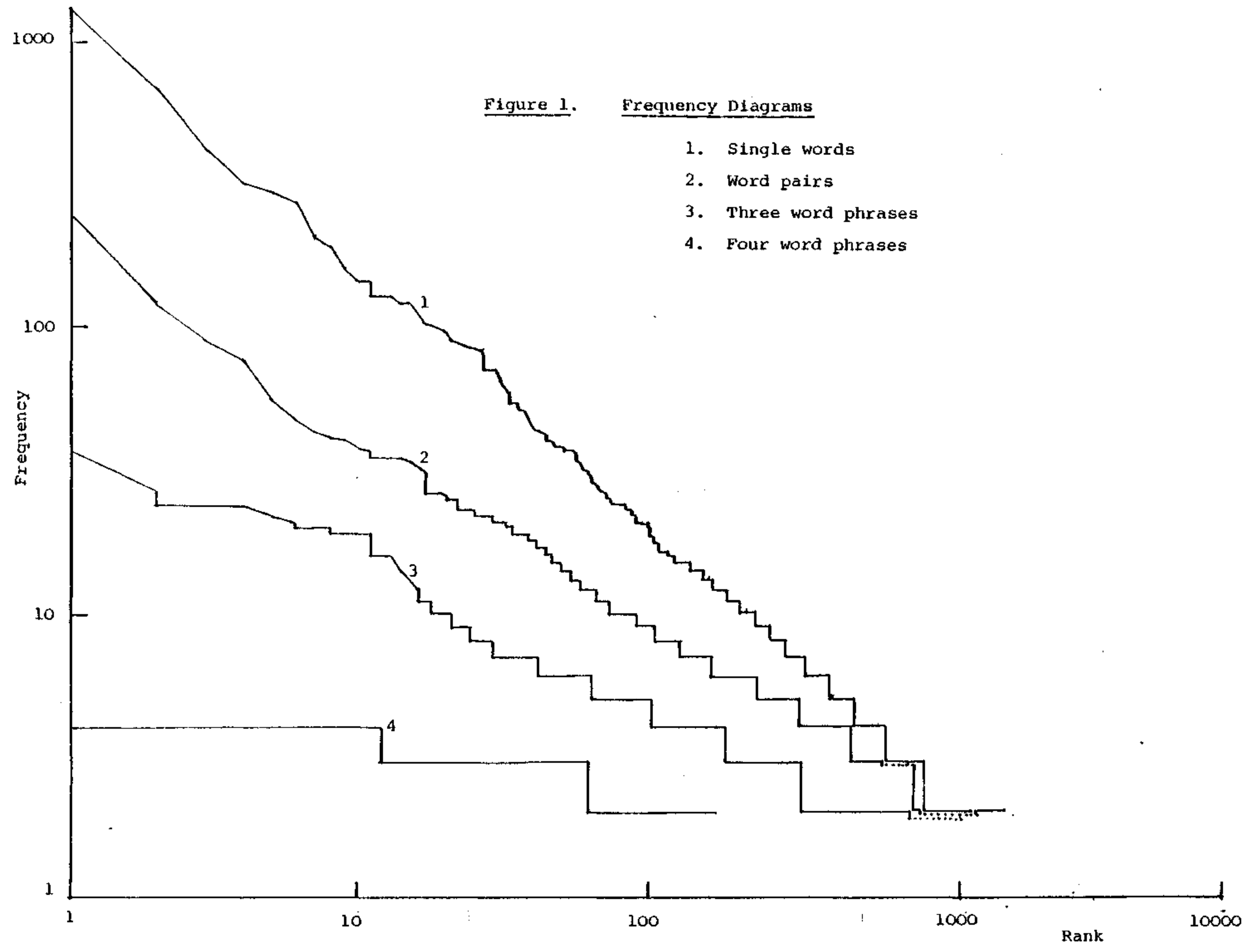

Figure 1. Frequency Diagrams

1. Single words
2. Word pairs
3. Three word phrases
4. Four word phrases

The unexpected result in Figure 1 is that the curves appear not to cross. However, we believe this is only because the text size is not large enough. The figure shows that $f_2 < f_1$ for all r less than about 1000. This should be an invariant for any size of text: the curves will cross at about r = 1000. For our sample text the total number of distinct word pairs with frequencies greater than 1 or 2 is similar to the number of words with higher frequencies and since it is only these frequently occurring pairs which are of primary interest, their number is fortunately relatively small.

We can compute these numbers more precisely from the Zipfian laws we derived empirically above. Let us consider a large text made up of T tokens and N different word pairs. For the last pair with rank N, the frequency computed from equation (3.2) must be just greater than $\frac{1}{2}$. So we put

$$\frac{K_2}{N^\beta} = \frac{1}{2} \tag{3.3}$$

The total number of tokens T can be computed by integration

$$T = \int_{\frac{1}{2}}^{N} \frac{K_2}{r^\beta}\, dr \approx \frac{K_2}{(1-\beta)} \left[N^{1-\beta} - \left(\tfrac{1}{2}\right)^{1-\beta}\right] \tag{3.4}$$

after substitution from equation (3.3) and neglecting the second term we get

$$T = \frac{1}{2}\,\frac{N}{(1-\beta)} \qquad \text{for } \beta < 1. \tag{3.5}$$

This predicts that the number of different word-pairs N increases in direct proportion to the length of text, unlike the number of single word-types.

Using the value $\beta = 0.65$ obtained from our sample text then

$$N = 0.7\ T. \tag{3.6}$$

In our sample T = 16,050; so from formula (3.6) we would expect the number of distinct word pairs to be approximately

$$N \approx 11{,}235$$

based on the observed values of β and T. Using the observed values of K_2 and β in equation (3.3) yields N = 10,073.

We counted

$$N = 10{,}635$$

which shows the theory is approximately correct.

Most of these pairs occur only once or twice. The number of distinct pairs, N_1, which have frequency greater than one can be computed by noting that N_1 is the largest rank for a pair with frequency greater than 1, so the computed frequency of this pair will be just greater than $\frac{3}{2}$. So

$$\frac{K_2}{N_1^{\beta}} = \frac{3}{2}$$

It follows, using equation (3.3) that

$$N_1 = N/(3^{\frac{1}{\beta}}) \qquad (3.7)$$

$$= \frac{N}{5.42} \text{ if } \beta = 0.65. \qquad (3.8)$$

Thus less than one-fifth of all distinct pairs have more than one occurrence in any text. For our sample text we compute the number with higher frequencies to be

$$N_1 = \frac{10635}{5.42} = 1962.$$

A count showed that

$$N_1 = 1742$$

which again gives us confidence that the theory is approximately correct.

The number with frequency 2, we counted to be 911; so the number with frequency greater than 2, N_2, is only

$$N_2 = 831.$$

This is less than 8% of the total number of distinct pairs or less than 6% of the total size of the text:

$$N_2 = 0.06\ T.$$

It is these pairs in which we are primarily interested.

For a long text such as James Joyce's Ulysses, which according to Zipf (1949) has 260,430 tokens and 29,899 words, we then expect the number of pairs occurring more than twice to be about

$$N_2 = 15{,}600.$$

which is just over half the number of words used in the text. The total number of all pairs in the text would be much larger, about

$$N = 180{,}000$$

Now the maximum possible number of pairs in a text of length T must equal the number of tokens of word pairs, T-1, i.e. 260,429 which is not much greater than the actual number of different pairs 180,000. This suggests that the total number of pairs possible in English is far above this number though we expect much less than the 900 million combinations which are theoretically possible with Joyce's word list of almost 30,000 words.

However, of more interest than these conjectures is that the total number of word pairs occurring frequently (i.e. more than twice) in a text such as Ulysses is about half in number the individual words in the text and is therefore not large and is manageable.

3.2 Storage.

Hashing. - The method of storage is similar to the hashing method recommended for single words. A hash number is generated for each of the two words, h_1 and h_2, and a new hash number generated by some arithmetic operation on these two. The sum $h_1 + h_2$ is the simplest suitable operation. So the address of the block on disc or cell in the central store would be

$$h = (h_1 + h_2) \bmod B \qquad (3.9)$$

where B is the maximum number of blocks (or cells) used to store the pairs. The word pairs can then be stored in the block in order of their frequency.

This method would be the same whether we were storing 10,000 frequent word pairs or all pairs numbered in millions. The hashing algorithm should be equally efficient in both cases.

List structure. - It would also be possible to hash only on the first word of each pair and then store a list of word pairs beginning with the most frequent. This list structure is convenient for many purposes; it is therefore convenient to add it to the hashing structure mentioned above using pointers.

4. Multi-Words

4.1 Frequencies. - We have computed the frequencies of all 3-word phrases and of the most frequent 4-word phrases for our sample legal text and the Zipfian distributions are shown in Figure 1. The frequencies of 3-word phrases, f_3, are small compared with the frequencies of word pairs, so the statistical evidence is less clear than in the previous case. Yet the frequencies obviously follow a law similar to the law for pairs in equation (3.2):

$$f_3 = \frac{K_3}{r^\gamma} \qquad (4.1)$$

where the constant slope γ and constant K_3 are given approximately by

$$\gamma = 0.505 \ , \ K_3 = 50 \ . \qquad (4.2)$$

and the invariant ratio $K_3/K_2 = \frac{1}{4}$.

The number of three word phrases which occur at least twice is given by the equation

$$\frac{K_3}{N_1^\gamma} = \frac{3}{2} .$$

$$\text{So} \qquad N_1 = (\tfrac{2}{3} K_3)^{\frac{1}{\gamma}} = 1036 \qquad (4.3)$$

We counted N_1 = 1228 which is in reasonable accord with the theory.

The number with frequencies greater than 2 (those in which we are most interested) we counted to be N_2 = 350. This is half the number of pairs with the same high frequencies. The corresponding number of 4-word phrases is 62, much smaller still.

About 90% of 3-word combinations in the text occur only once and the meaning of each is made up of two or three parts, so there is no need to store them.

Each 3-word phrase contains within it two word pairs, i.e.

ABC includes AB and BC

Thus the frequencies of AB and BC should be reduced by 1 if ABC is stored. However, an exception occurs each time ABC is part of a frequent 4-word phrase. The four letter combination

ABCD includes ABC and BCD

and both ABC and BCD include the pair BC; however, the frequency of BC should obviously not be reduced by 2 because it occurs within ABC and BCD. It should be reduced only by 1. This difficulty is overcome by storing information on 4, 5, 6, etc. word phrases until all have a low frequency. Then if ABCD is not part of a higher order phrase with frequency greater than 2 we can reduce the frequencies of ABC, BCD, AB, BC, CD and possibly of A, B, C and D by 1 for each occurrence of ABCD. This has not been done in Figure 1.

4.2 Storage. - Multi-word phrases can be stored by hashing. If h_1, h_2, etc. are the individual hash numbers for the individual words the address of the block (or cell) can be given as

$$h = (h_1 + h_2 + \dots) \bmod B$$

Alternatively, or in addition a tree structure can be used with pointers from each word record to a list of pair records ordered according to their frequencies; and each pair record pointing to a list of 3-word phrases (when appropriate) and so on. This is the structure used in our current computer analysis; it is illustrated in Figure 2.

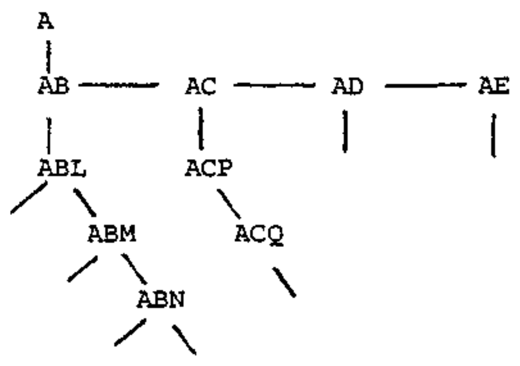

Figure 2. Tree structure for storage of word pairs AB, AC, etc. and multi-word phrases ABL, ABM, etc.

5. Conclusion

We have discussed methods of structuring dictionary files which were developed in the context of information retrieval. We recommend the use of division hashing, with an adequate block size (at least 10 records), and a good method of handling overflow, such as the linear quotient method. These techniques ought to be equally applicable to dictionary files used in machine translation.

We then addressed the question of whether dictionary records should be created for pairs of words and for larger word groupings. Frequency counts based on a corpus of text suggest that the number of such groupings with frequencies above one or two is surprisingly small (of the same order of magnitude as the number of different words), and this is reinforced by theoretical considerations based on Zipf's law. These groups can also be stored and retrieved quickly by hashing techniques.

It is interesting to examine the implications of these results for larger text corpora and even for the totality of an individual's linguistic input, which over a period of 20 years might amount to 1,000 million words of spoken and written material. There is little justification for extrapolating into these regions because of the effect of the finiteness of the supply of different single words. Nevertheless, if we do so we find that our observed value of β for word-pairs predicts that out of 700 million different pairs which will occur, only some 200,000 pairs will occur more than 100 times (representing 2% of pair tokens), and as few as 6,000 pairs will occur 1,000 times or more (0.6% of pair tokens).

If 100 repetitions is accepted as a suitable threshold of reinforcement, above which a word-pair may be remembered as a unit, we may tentatively suggest a figure of 200,000 for the number of word-pairs that should be stored in a dictionary to imitate human memory.

References.

J.R. Bell and C.H. Kaman, "The Linear Quotient Hash Code", Comm. A.C.M., 13, 675-677, 1970.

K. Devine and F.J. Smith, "Direct file organization for lemmatized text retrieval", Information Technology, 3, 25-32, 1984.

J. Fedorowicz, "The Theoretical Foundation of Zipf's Law and its Application to the Bibliographic Database Environment", J.Am.Soc.Inf.Sci., 33, 285-293, 1982.

L.D. Higgins and F.J. Smith, "On-line Subject Indexing and Retrieval", Program, 3, 147-156, 1969.

J.Q. Jamison, "Studies in Information Processing", Ph.D. Thesis, The Queen's University of Belfast, N. Ireland, 1977.

P. Larson, "Analysis of Uniform Hashing", Journal A.C.M., 30, 805-819, 1983.

J. Martin, "Computer Data-Base Organization", Prentice-Hall, N.J., 1975.

W.D. Maurer, "An Improved Hash Code for Scatter Storage", Comm. A.C.M., 11, 35-38, 1968.

T. Ore, "Structured Requirements to a Free-Text Retrieval System", paper presented at a Symposium on Legal Information Retrieval in Europe, Strasbourg, June, 1979.

W.W. Peterson, "Addressing for Random-Access Storage", IBM J. Research and Development, 1, 130-146, 1957.

G.K. Zipf, "Human behaviour and the principle of least effort", Addison-Wesley, 1949.