EFFICIENT CODING OF SPEECH SYNTHESIS DATA

by

Simin Hosne-Sanaye

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL ENGINEERING

In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

In the Graduate College

UNIVERSITY OF ARIZONA

1 9 8 4

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Simin Hosne-Sanaye

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

L.C. Schooley for J. Parry — August 3, 1984

J. PARRY
Associate Professor of Electrical
and Computer Engineering

Date

ACKNOWLEDGMENTS

The author would like to express her appreciation and special recognition to Dr. J. Parry, her major professor and director of the thesis.

Recognition is also extended to Dr. L. Schooley and Dr. R. Strickland for their helpful reviews of this study.

Finally, the author wishes to extend her personal gratitude to her husband, Aziz, and her son, Sepand, whose unlimited love and support were important factors in completing this study.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABSTRACT

The Shannon-Fano coding algorithm was used to obtain an upper bound to the average code length to speech synthesis data. The objectives of the study were (1) to outline steps necessary toward applying a variable code length technique for encoding speech synthesis data, and (2) to apply these steps to existing data to study the savings in computer space requirements.

In the case study, probabilities of source alphabet letters were estimated and a biasing factor, λ, was applied to eliminate zero probabilities for unobserved letters. The resulting probabilities were used to generate bounds on the average code length as a function of a cutoff value c. The average code lengths were used to create charts of storage requirements for an application of the variable length code technique. Also, the probabilities were used to estimate total code lengths for the case study. The results suggested that for a certain range of λ and c, the storage requirements are substantially less than that used for a fixed length code.

INTRODUCTION

A TMS 5220 voice synthesizer is used in the Electrical Engineering Speech Synthesis Laboratory at the University of Arizona. For this voice synthesizer, the speech data are compressed using a linear predictive coding.[1] The compressed data are in the form of 39 bits per 50 ms time frame. TMS 5220 accepts sequences of these 39 bit blocks as input and, in turn, changes them into the synthetic speech wave form. Ordinarily, the speech data coded with a linear predictive coding algorithm are stored on diskettes and/or computer memory for future use. The memory storage required for storing the coded data as a function of the coding algorithm (fixed 39 bits vs. variable length code) is studied in this thesis. Only voiced segments of speech were studied here.

Problem Definition

Some 213K bits of memory are required to store 2 minutes of speech data in one commercial product which uses the TMS 5220 voice synthesizer. The parameters used by the Speech Production model in the TMS 5220 are transmitted to the integrated circuit as a 39 bit string for every 50 ms time frame (fewer bits are required if silence or unvoiced speech is to be produced during the frame). Various coding

---

1. TMS 5200 Voice Synthesis Processor Data Manual, Preliminary. October 1980, Texas Instruments, Inc.

algorithms can be used to optimize the utilization of the computer memory. One possibility is to use a variable length coding to associate shorter codes to more probable parameter sets.

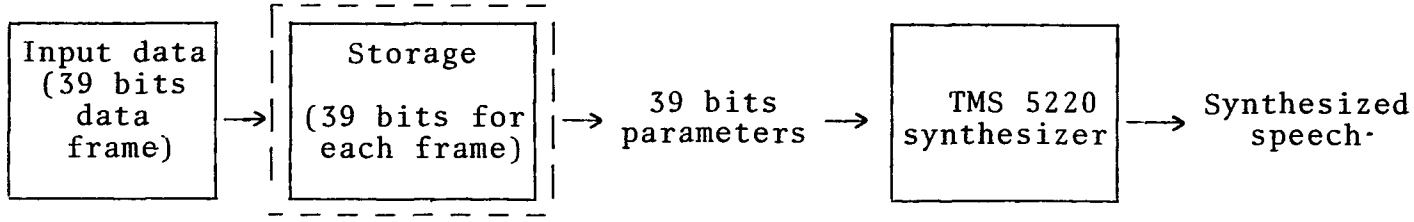
The objectives of this thesis are to outline the steps necessary to study the feasibility of applying a variable length coding technique to the speech data presently used in one application ROM for the TMS 5220. The present storage system for speech data and an alternative storage system are shown schematically in Figure 1.

To study the feasibility of applying any variable length coding technique, the following major problems are anticipated:

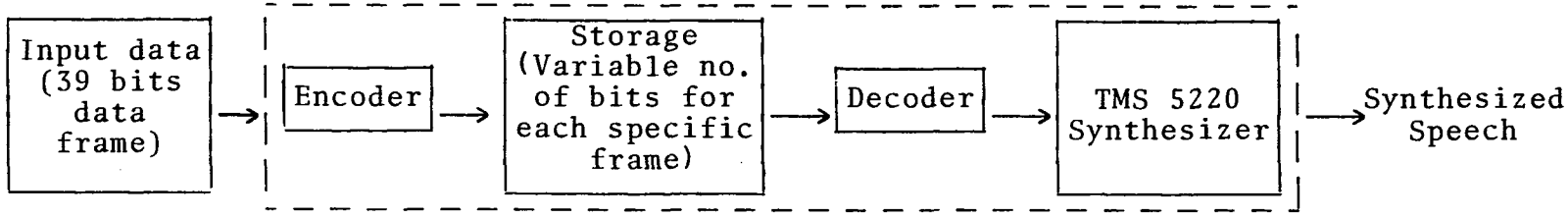
1. Acquiring statistically sufficient data to estimate the probability of occurrence of each specific speech frame.
2. Developing computer programs to calculate the estimated probabilities for large data sets.
3. Estimating an upper bound for the average code length using a variable code length technique.
4. Computing the amount of required storage memory.

Literature Review

Morse's method of telegraphy (1832) with its assignment of short patterns to frequently occurring letters is mentioned in the information theory history [14] as one of the first efficient coding techniques. Claude Elwood Shannon was the first to model the message source as a stochastic process, in 1948 [14]. "The coding theory was born and nurtured in the 1950's, was well past the crawling stage when the 1960's ended" [15]. Shannon and Huffman were the main contributors



a.



b.

Figure 1. Pictorial representation of the storage systems for speech data. -- (a) Present storage system; and (b) Alternative storage system.

to the area of source coding in the 1950's. Another important related event in the 1950's was recognition of Shannon's work by Soviet mathematicians [15, 7]. By the end of the sixties, the field of information theory was well defined. "Central, of course, is the Shannon Theory with its direct and immediate preoccupation with source and channel coding. All of probability theory, stochastic processes, and mathematical statistics, including decision and estimation theory, are fundamental, and in large part prerequisite to an understanding and appreciation of information theory" [17]. The central theme of information theory follows what Shannon initially proposed:

i) discrete source coding,

ii) coding for a noisy channel,

iii) source coding with a fidelity criterion [17].

The definition of entropy had already resolved the problem of discrete source coding in its most basic form. In the fifties and sixties, more source coding algorithms appeared. Today, the emphasis in the area of discrete source coding is on further refinements and applications [17].

The objective of using Shannon's ideas in engineering applications was to improve performance and at the same time to minimize the extra cost and added complexity. Applications in space communications and high-density storage of data in computers have proved successful in achieving the objective [19].

"There are two well known approaches to the problem of coding a source so that the code can be decoded uniquely [1, Ch. 3]. One

approach uses fixed-length codes and makes use of asymptotic equipartition property. The other uses variable-length codes and minimizes mean length, subject to the constraint that the lengths satisfy the Kraft inequality" [3]. Campbell [3] proves that for a finite discrete source the average code length is bounded from below by entropy.

In general, the encoding of a source is dependent on the statistical parameters of the source. Research has been concentrated on employing "universal" codes that are independent of the source statistics [5,6,20,21]. For these codes, it can be shown that the asymptotic behavior approaches codes which are based on known statistics.

Gilbert, on the other hand, treats the source statistical parameters as inaccurate, rather than unknown, and introduces coding procedures that anticipate some worst error estimation of these parameters [9]. He estimates the source letter probability by observed frequencies and a biasing parameter.

A series of papers [4,5,12,13] deals with the optimization of computer memory requirements for Question-Answer (QA) systems. In these systems, a rate distortion function is defined as the minimum storage space required to guarantee a certain quality of responses.

Pearl [12] shows that memory saving can be achieved by retaining precise true values for a certain fraction of propositions on the original list and ignoring the rest. During the answering phase, the former is answered with certainty (P = 0,1) and the latter with

uncertainty (P = 1/2). Such a coding scheme yields bounds on the storage space required.

Later works on QA systems treat the problem in a more general manner based on general system parameters such as input statistics [4,5].

In this study, a discrete source coding, namely Shannon-Fano encoding, is used to obtain an upper bound on the average code length. This coding is a variable-length coding technique.

The probabilities of the source alphabet are estimated using the Gilbert biasing approach [9]. In this approach statistics about the source alphabet are used to obtain a more efficient coding. Also, a biasing factor, λ, is used to guard against data samples that might have grossly different statistics from those that were used for design of the encoding algorithm.

PROPOSED SOLUTION

To study the feasibility of applying a variable code length technique to the speech data required for TMS 5220, a reliable measurement of the probabilities of the alphabet letters is required. Each letter corresponds to a specific speech frame. The probability of occurrence of each letter of the alphabet with some confidence level can be measured if sufficient sample sizes are available. These probabilities then can be used to determine the average code length, entropy, and individual code lengths. For this study, the source alphabet is $S = \{L_1, L_2, \ldots L_N\}$ where $N = 2^{39}$ and L_i, the ith letter of the alphabet, is a 39-bit-long string of binary numbers referred to as a frame. The code alphabet is the binary set $\{0,1\}$. The estimated probability of occurrence of the ith letter in the source alphabet is denoted by q_i and the true probability of the ith letter is denoted by p_i, for $i = 1, 2, \ldots, N$. For both probabilities $\sum_{i=1}^{N} q_i = \sum_{i=1}^{N} p_i = 1$.

For a small size source alphabet, such as a teletype with 32 characters, the task of getting enough data points and high confidence interval is rather simple.

When the alphabet size is large (e.g. 2^{39}), the probability calculations using conventional methods, such as storing a count for each alphabet letter may be prohibitive. In addition, for any extension of the alphabet, the possible letter combinations increase

rapidly. As a result, any computer program designed to assess these probabilities must be capable of overcoming the storage problems.

To assure that the estimation errors for the calculated probabilities are small, the following steps are suggested:

1. Establish a programming procedure to collect the frequencies of all the letters in a sample data set.
2. Find acceptable number of representative data sets necessary to obtain reasonable confidence intervals for the estimated probabilities.

Computer programs for calculating the probabilities of all the letters in a sample data set written in PASCAL and the Motorola 68000 assembly language will be discussed in more detail. The confidence intervals on calculated probabilities will also be discussed. As an alternative and/or complement to this probability estimation, a biasing factor, λ, is introduced to modify the probability estimates so that the unobserved events will have nonzero probability estimates [9].

Finally, after the probabilities are estimated, a bound on the average code length (l_{ave}) is calculated. If the bound for l_{ave} is less than 39 bits, a variable-length coding technique may be used for encoding and decoding the fixed codes.

Computer Programming Requirements

The computer storage and time requirements to calculate the frequencies of all the alphabet letters will be large if the alphabet size is large. Any extension of the alphabet will require even more computer memory and time. When dealing with speech data and large

alphabet size (2^{39} in our case), not all the alphabet letters will occur with equal probabilities. As the size of the alphabet increases, the number of letters with a very small probability of occurrence will also increase.

A programming method for calculating the estimated probabilities $q_i = \text{prob}(L_i)$ for i = 1, 2, . . . , N can be summarized as follows:

1. A sorting algorithm sorts the sample data in a way that same letters will be placed next to one another. This sorting is required to decrease the number of comparisons. For a sample size of T, a straightforward counting of occurrences will result in T2 comparisons as opposed to T x $\log_2$ T comparisons in the case of QUICKSORT [16].
2. A second algorithm eliminates the redundant letters and, at the same time, asigns a count number for each letter. This count number is the frequency of occurrence of that specific letter.
3. Entropy, average code length, and probability of occurrence of each letter are estimated from the observed frequencies.

For studying the conditional occurrence of letters (extensions of alphabet), the adjacent letters in the data can be combined via a simple change to a parameter value in the computer program to form longer letters.

Note that this method is are based on the assumption that many of the 2^{39} letters do not appear in the sample data. This is not an unreasonable assumption since the speech coefficients represent a

series of cross-sectional areas for the vocal tract and it cannot assume all the 2^{39} different shapes. By far the most important and time sensitive piece of the program is the search and sort work (i.e. to search for identical letters and to sort them so they are adjacent). A sorting algorithm called QUICKSORT [16] is used, which takes $T \times \log_2 T$ comparisons to sort the data, when T is the sample size. For alphabet extension studies only a minor change of parameter in the program is necessary. Another program called UNIQUE [11], written in PASCAL, strips all the redundant data frames from the output of QUICKSORT and calculates a frequency number for each distinct frame.

Probabilities and Confidence Intervals

The problems of finding confidence intervals for each probability are two-fold. The first problem is to have independent and representative speech data. The second problem is to have a sufficient number of data sets with the above qualifications of speech data.

Of the above problems, the first one becomes especially complicated for the large size alphabet sets such as in the case study here. The complications are due to the fact that a decision must be made as to what makes the speech data representative and unbiased. Some of the factors to be considered are the language (e.g., English), the source of speech (e.g., male, female, adult, etc.), the text spoken, etc. The answers to these questions are beyond the scope of this study. For the purposes of this thesis, it is assumed that few minutes of speech from an adult, English-speaking male is representative of the data samples used for TMS 5220 applications. The second problem is associated with

the estimation of all confidence intervals. Any sample speech data, with a large alphabet size, will contain many letters with frequency of one or zero. These letters might vary from one sample to another. The letters that have high probabilities, however, will be frequent for most sample data sets. Based on this, it can be argued that only the more frequent (common) letters have to have a high confidence associated with them. The procedure outlined below includes all the alphabet letters; but for application purposes, only the letters with relatively large probability estimates are considered. A confidence interval on each probability can be calculated as outlined in the following steps [10].

1. Denote the probability estimate of letter i (L_i) in data set j by q_{ij} where i = 1, 2, . . . , N and j = 1, 2, . . . , j_{max}. N is the source alphabet size and j_{max} is the maximum number of data sets available. This estimate is represented as:

$$q_{ij} = \text{estimated prob}(L_i) = t_{ij}/T_j$$

where T_j is the size of the jth sample data and t_{ij} is the number of observations of L_i in jth sample data set. Obviously, any unobserved L_i has the estimated probability $q_{ij} = 0$.

2. Repeat step one for all the repetitions of data sets available.
3. Calculate the mean for parameter q_{ij} = j = 1, 2, . . . j_{max}.

Denote the population mean by μ_i and sample mean by $\overline{X}_i$,

$$\bar{X}_i = \left(\sum_{j=1}^{j_{max}} t_{ij}/T_j \right)/j_{max}.$$

4. Calculate the sample variance by

$$S_i = \sum_{j=1}^{j_{max}} (t_{ij}/T_j - \bar{X}_i)^2/(j_{max} - 1).$$

5. Assuming the mean is normally distributed and the variance is Chi square, then $[(\bar{X}_i - \mu_i)\sqrt{j_{max}}]/s_i$ is distributed according to a student t distribution [10].

6. $(\bar{X} - t_{1-\alpha/2} (S_i/\sqrt{j_{max}}), X_i + t_{1-\alpha/2} (S_i/\sqrt{j_{max}})$ is a confidence interval for μ_i. Calculae this interval using the appropriate α level and a t table and the degrees of freedom $j_{max} - 1$. This $1 - \alpha/2$ is actually a confidence level on q_i.

The assumption of unobserved events having a probability of zero could result in poor coding because it could be drastically inefficient on some other source of sample input. For example, this assumption can result in an infinite code length using the Shannon-Fano algorithm [$l_i \geq \log_2(1/q_i)$], where l_i is the code length associated with frame L_i. A non-zero estimate of unobserved letters reduces the chance of underestimating these probabilities, consequently reducing the average code length for probability distributions close to, but not equal to, the estimate with many zeros. Gilbert [9] suggests the estimate:

$$q_i = (t_i + \lambda)/(T + N \times \lambda) \quad \text{for } i = 1, 2, \ldots, N$$

where N is the alphabet size (2^{39} in our case), T, is the sample size,

t_i is the frequency of letter L_i, and λ is a positive number. Therefore, all probabilities are biased toward 1/N and no zero probabilities exist. This can be interpreted as adding λ extra letters of each kind to the sample data on hand. For large values of λ, in this case $\lambda >> T/N$, q_i's are nearly equal. For $\lambda = 0$, $q_i = t_i/T$. The estimated q_i has a high probability of being close to the real probability if $T/(N \times \lambda)$ is large ($T/(N \times \lambda) >> 1$) [9]. Note that

$$\sum_{i=1}^{N} q_i = \left[\sum_{i=1}^{N} (t_i + \lambda)/(T + N \times \lambda) \right] = (T + N \times \lambda)/(T + N \times \lambda) = 1.$$

Average Code Length

In the previous section, probability estimates (q_i's) were derived and can be used in analysis and study of average code length. Given source symbols and their corresponding probability estimates, q_i for $i = 1, 2, \ldots, N$, the Shannon-Fano coding algorithm [1] states that for each q_i an integer l_i exists such that:

$$\log_r(1/q_i) \leq l_i < \log_r(1/q_i) + 1$$

where r is the number of letters in code alphabet (here r = 2), and l_i is the code length and l_i's satisfy the Kraft inequality

$$\left(\sum_{i=1}^{N} 2^{-l_i} \leq 1 \right)$$ [18]. This indicates that there exists a uniquely decodable binary code for:

$$S = \{L_1, L_2, \ldots, L_N\}$$

with

$$q_1 \geq q_2 \geq \cdots \geq q_N.$$

where q_k = prob(L_k) and k = 1, 2, . . . , N [1].

Code lengths derived from the Shannon-Fano coding algorithm may be less efficient than those derived from Huffman coding [1]. However, the code lengths can be derived independently of each other using the Shannon-Fano algorithm. Also, the upper bound on l_{ave} for the Shannon-Fano algorithm is certainly an upper bound for Huffman coding, i.e.:

$$l_{ave}(\text{Huffman}) \leq l_{ave}(\text{Shannon-Fano});$$

where l_{ave} is the average code length defined by $l_{ave} = \sum_{i=1}^{N} q_i \times l_i$. It can also be shown that [1]:

$$H(S) \leq l_{ave}(\text{Huffman}) \leq l_{ave}(\text{Shannon-Fano}) \leq H(S) + 1;$$

where H(S) is the entropy.

Entropy is defined by:

$$H(S) = \sum_{i=1}^{N} p_i \log_2(1/p_i),$$

where p_i is the true probability of L_i [1]. If the q_i's were correct, then the Shannon-Fano algorithm would produce codes with average length l_{ave}, with limiting l_{ave} approaching H(S) from above. If the q_i's differ from the actual probabilities, then l_{ave} will approach values greater than H(S) [9].

The l_{ave} is a function of the q_i's and therefore is a function of both the sample size, T, and λ. For this study, T is constant and l_{ave} is studied as a function of λ. The upper bound, i.e.:

$$l_{ave} \leq \left[\sum_{i=1}^{N} q_i \times \log_2(1/q_i) \right] + 1,$$

is compared with a fixed code length of 39.

Before applying the q_i's from the previous section to this upper bound, another bound on some q_i's should be discussed. The probability estimates $q_i = t_i/T$ may be divided according to the following convention:

$$q_i = \begin{cases} (t_i + \lambda)/(T + \lambda \times N) \text{ if probability estimate of } L_i \geq c \\ p_o \text{ otherwise} \end{cases}$$

Let L_m for $m = 1, 2, \ldots, M$ be the distinct alphabet letters whose probability estimates are greater than a cutoff value c, and

$$p_o = (1 - \sum_{m=1}^{M} q_m)/(N - M).$$

In this case we still have $\sum_{i=1}^{N} q_i = 1$ and can rewrite the upper bound on l_{ave} as below:

$$l_{ave} < \left[\sum_{i=1}^{N} q_i \times \log_2(1/q_i) \right] + 1$$

$$= \sum_{m=1}^{M} q_m \times \log_2(1/q_m) + \left[\sum_{\substack{j=1 \\ \text{and } L_j \neq L_m \\ \text{for } m = 1, 2, \ldots, M}}^{N} p_o \times \log_2(1/p_o) \right] + 1$$

$$= \sum_{m=1}^{M} q_m \times \log_2(1/q_m) + \log_2(1/p_o) \left[\sum_{\substack{j=1 \\ \text{and } L_j \neq L_m \\ \text{for } m = 1, 2, \ldots, M}}^{N} p_o \right] + 1$$

$$= \sum_{m=1}^{M} q_m \times \log_2(1/q_m) + \log_2(1/p_o)(1 - \sum_{m=1}^{M} q_m) + 1.$$

$\text{Log}_2(1/p_o)$ can be interpreted as the code length for non-frequent events. Denote this length by l_s. Rewriting the above relation:

$$l_{ave} < \sum_{m=1}^{M} q_m \log(1/q_m) + l_s(1 - \sum_{m=1}^{M} q_m) + 1.$$

Parameters c and λ contribute to the variability of l_{ave}. Two approaches are used here to establish upper bounds on l_s. In the first approach, it can be argued that if a fixed code length was used, 39 bits would be enough to encode all the source alphabets (i.e. in the worst case all the probabilities are 1/N). If the code length associated with the smallest probability, $q_{min} \geq c$, is denoted by l_{qmin} and the code length associated with c is l_c, then all the remaining improbable letters can be coded by using a constant code length of $l_{sl} = l_{qmin} + 39$ (for Shannon-Fano algorithm), l_{sl} is the code length for non-frequent events using the first approach. However

$$lq_{min} < l_c < \log_2(1/c) + 1,$$

therefore:

$$l_{sl} < \log_2(1/c) + 40.$$

Thus, the following can be written:

$$l_{ave} < \sum_{m=1}^{M} q_m \log_2(1/q_m) + l_{sl}(1 - \sum_{m=1}^{M} q_m) + 1;$$

$$l_{ave} < \sum_{m=1}^{M} q_m \times \log_2(1/q_m) + (\log_2(1/c) + 40)(1 - \sum_{m=1}^{M} q_m) + 1.$$

In the second approach, it can be claimed that 39 bits will always be enough to encode any uncommon event, if enough bits were available initially to identify the event that an uncommon letter has happened. Using Huffman's approach, we can state that

$$\lceil \log_2[1/(1 - \sum_{m=1}^{M} q_m)] \rceil$$

bits are necessary to indicate that an uncommon event has occurred. Thus the code length for a non-frequent event using the second approach, l_{s2}, can be expressed as:

$$l_{s2} = \log_2[1/(1 - \sum_{m=1}^{M} q_m)] + 39,$$

and

$$l_{ave} < \sum_{m=1}^{M} q_m \times \log_2(1/q_m) + l_{s2}(1 - \sum_{m=1}^{M} q_m) + 1,$$

thus

$$l_{ave} < \sum_{m=1}^{M} q_m \times \log_2(1/q_m) + (\log_2[1/(1 - \sum_{m=1}^{M} q_m)] + 39)$$
$$(1 - \sum_{m=1}^{M} q_m) + 1.$$

Both l_{s1} and l_{s2} and their effects on the l_{ave} are studied as functions of λ and c. For c and λ values that result $(\log_2(1/c) + 40) > [\log_2 1/(1 - \sum_{m=1}^{M} q_m) + 39]$ the second approach will result in more efficient coding (for a given data set). Of course, if the above inequality is reversed l_{s1} would result in shorter l_{ave}.

In general c is larger than $(1 - \sum_{m=1}^{M} q_m)$ and it may be conjectured that l_{s2} is always superior to l_{s1} and will result in a shorter code length for uncommon letters. The proof of this, however, is beyond the scope of this study.

Storage Requirements

Some additional hardware and/or storage would be required to implement any variable code length technique on a TMS 5220. Figure 2 summarizes these additional requirements.

An encoder is necessary to encode the received 39 bit inputs into a variable length code. Also, a decoder is required to change the coded letters back into the 39 bit fixed length code for input to TMS 5220. This decoder can be set up so that the common letters are decoded using a table look up procedure (M x 39 bits of memory needed). The uncommon letters, however, do not need a table look up, since if the first few bits that indicate an uncommon event has happened are disregarded, the rest of the code is the actual 39 bit frame ready for use by TMS 5220. Aside from the addition of the encoder/decoder logic to the present system and the table look up, the memory space requirements of the present system vs. the proposed system may be compared. To store a sample data of size T in the present system, T x 39 bits of memory are required. For a variable code length encoding scheme this memory requirement is approximately T x l_{ave}.

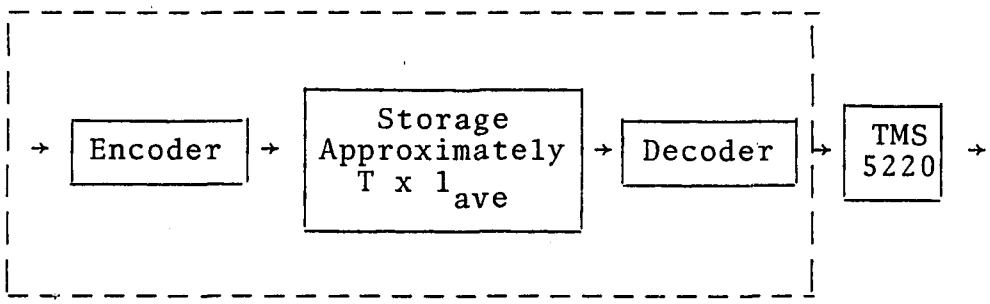


Figure 2. Additional hardware and memory requirements for implementing a variable length code technique.

CASE STUDY

This study is concerned with implementation of the proposed variable code length technique analysis on a given set of speech data. This data set consists of 7766 speech frames which represent a few minutes of real speech in English. Each frame consists of 12 parameters, 10 of which are coded reflection coefficients of a lattice filter vocal tract model with the other two pitch and energy. Of these frames, 5464 are considered voiced and the rest are unvoiced. A voiced speech frame is defined as a frame for which neither pitch nor energy are zero. We are only concerned with the analysis of the voiced data in this study. Here, a frame would refer to the ten k-parameters only.

Computer Programs

The collection of programs that is used to calculate the probability and entropy estimates consists of two parts. The first part, which is written in PASCAL, is called FREQAS. This program, in turn, is composed of a number of procedures. In the main program (FREQAS) the data frames for which either pitch or energy values are zero are eliminated. Table 1 lists the functionally important procedure names and their basic functions.

Each of the procedures may call other procedures. For a complete description of FREQAS, refer to the listing of the programs in the appendix.

Table 1. List of major procedures used in this study.

Procedure Name	Function
FREQAS	Main program. Eliminates unvoiced speech data frames, links to the MAINAS, and invokes the following three procedures.
1. QUICKSORT	Sorts the k-parameters according to their descending order and puts each datum on one line of output file.
2. UNIQUE	Strips adjacent duplicate lines. Inputs the output of QUICKSORT. Also calculates a frequency number for each distinct datum. Outputs the frequency number along with the k-parameters.
3. ENTROPY	Calculates probabilities and total ENTROPY of sample data. Gets its input from UNIQUE and outputs the entropy value on the display.
MAINAS	Stores all the voiced data frames and is called by the sorting procedures in FREQAS.

All the data frames are stored in the main memory of a Motorola 68000 mini-computer. The physical rearrangement of the frames in memory, needed for the sorting routine, is handled by the Motorola 68000 assembly language program. The program name is MAINAS and is called from the REARRANGE procedure by the QUICKSORT procedure. This part is written in assembly language to increase the memory available to the sorting program and to overcome the deficiency of the system in software tools availability (i.e. ability to physically reposition a data frame within a PASCAL file).

The PASCAL program needs storage for a maximum of two frames at a time, plus storage for stacking the REARRANGE pointers. The collection of FREQAS and MAINAS programs can accept up to 8000 voiced data frames. This limitation is imposed by the fact that PASCAL's stack and heap storage on Motorola 68000 is constant.[1] This constant is too small for the full execution of more than 8000 voiced data frames. All the interchange for sorting purposes is handled at assembly memory area and is PASCAL-independent. The execution of the MAINAS for 5464 frames takes 45 minutes on a Motorola 68000, operating with a 4 MHz clock.

High-level flow charts of FREQAS and MAINAS programs are presented in Figures 3 and 4.

Besides the above programs, other PASCAL programs are written to reduce and/or complete the results of the main programs. A list of

---

1. Motorola 68000 PASCAL Manual.

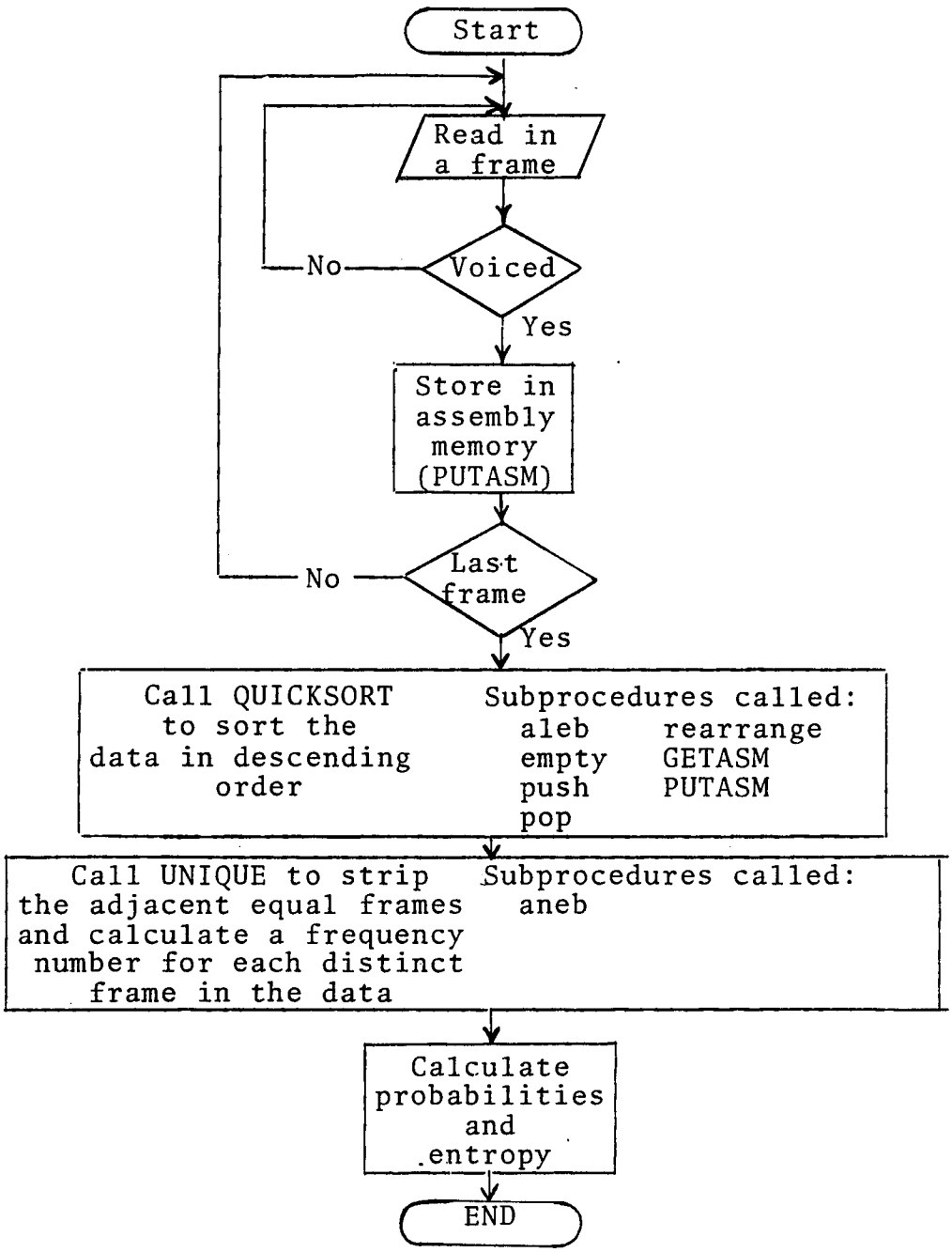


Figure 3. High-level flow chart of FREQAS program.

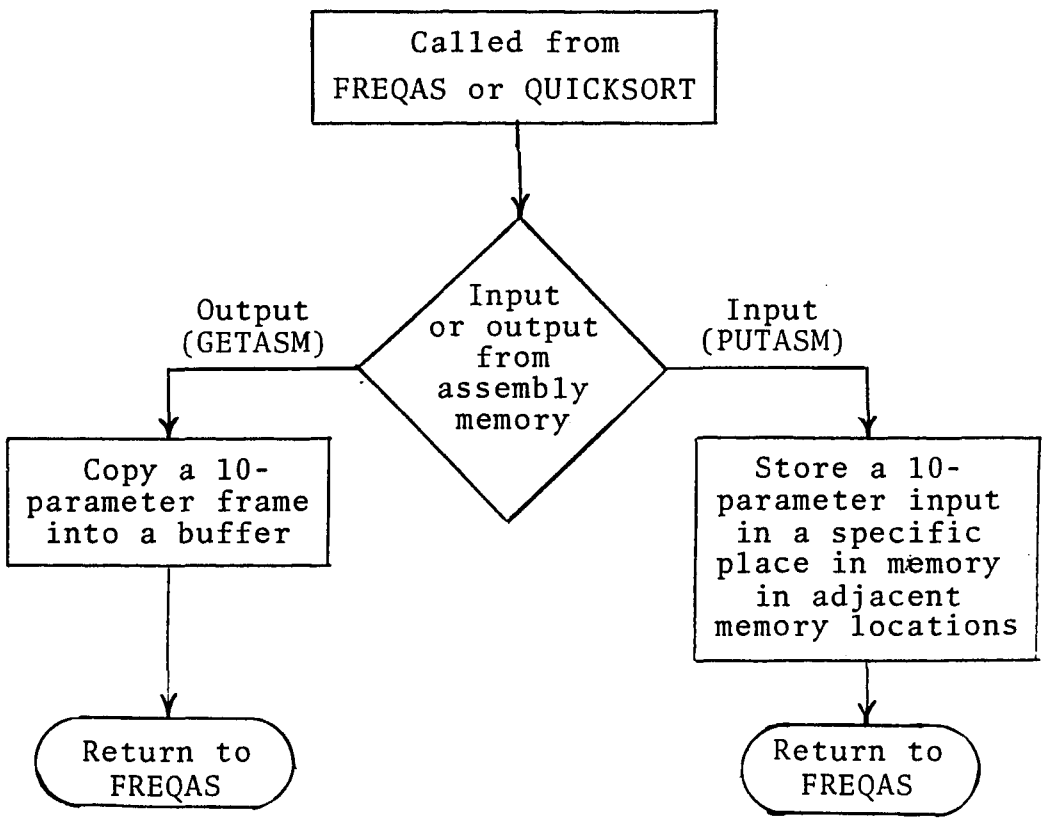


Figure 4. High-level flow chart of MAINAS program.

some of the more important of these programs, along with a description of their basic functions are summarized in Table 2.

The input to the main program (FREQAS) for this study were obtained from the Texas Instruments vocabulary ROM part number VM61002, which contained the word list shown in Table 3 spoken by an adult male English-speaker. Table 4 summarizes some results of the reduced data. In this table the number of frequency k frames refers to the number of distinct frames that their frequency of occurrence is k in the sample data.

The results of the other calculations and analyses will be presented later.

Refer to Summary and Conclusions section for suggestions on how to expand this program to handle more than 8000 frames.

Probabilities and Confidence Intervals

Enough sample data were not available for this study to calculate the confidence levels for each calculated probability. As an alternative, the possibility of dividing the existing data into several parts was considered. However, since the majority of the observations have a frequency of one, the idea was quickly discarded. As a result, these confidence levels cannot be carried over to the average code length measurements. However, the procedure suggested in the proposed solution is exercised for the common letter for which $q_{11} = 32/5464 = 0.006$. To do this we assume that $j_{max} = 5$, $q_{12} = .005$, $q_{13} = 0.05$, $q_{14} = .004$, and $q_{15} = .006$. For this scenario, the mean $X_1 = .0052$ and variance $S_1 = 2.25 \times 10^{-4}$ and $\alpha = 0.05$ and the degrees of freedom are

Table 2. List of some important support programs.

Program Name	Function
1. TRIM	This program inputs the output of UNIQUE and only retains the frequency numbers that are > 1.
2. LAVE	This program inputs the output of TRIM and a value and calculates the average code length for a range of c values where $q_i = (t_i + \lambda)/(T + N \times \lambda)$ when $t_i/T \geq c$.
3. LAVE1	This program inputs the output of TRIM and a λ value and calculates the average code lengths for a range of c values where $q_i = (t_i + \lambda)/(T + N \times \lambda)$ when $q_i \geq c$.
4. LCODE	This program calculates l_i's according to the Shannon-Fano coding with $l_s = l_{s1}$ and outputs the total storage requirements in bits.
5. LCODEH	This program calculates l_i's according to the Shannon-Fano coding with $l_s = l_{s2}$, and outputs the total storage requirements in bits.

Table 3. Vocabulary ROM for case study data.

LPC VOCABULARY DATA IN VM61002 (MALE SPEAKER NO. 5)

WORD PHRASE	LOOK-UP TABLE	START ADDR	WORD PHRASE	LOOK-UP TABLE	START ADDR	WORD PHRASE	LOOK-UP TABLE	START ADDR	WORD PHRASE	LOOK-UP TABLE	START ADDR
ZERO	0002	019E	HENRY	006A	0F39	START	0084	1326	ABOUT	009E	172B
ONE	000C	02FF	INDIA	00B8	1AE7	STOP	00D2	1F41	GAGE	00EC	22FB
TWO	0016	047C	JULIET	0106	2737	TIMER	0120	2BB2	GATE	013A	2F8B
THREE	0020	058A	KILO	0152	3324	VALVE	016C	376E	GET	0196	3B86
FOUR	0004	01EB	LIMA 1	006C	0F89	LINE	0086	1371	GO	00A0	1788
FIVE	000E	0342	MIKE	00BA	1B3B	MACHINE	00D4	1F70	GREEN	00EE	234A
SIX	0018	04B3	NOVEMBER	0108	279D	UP	0122	2BFD	HIGH	013C	2FD4
SEVEN	0022	0606	OSCAR	0154	3373	DOWN	016E	37CC	HOLD	0188	3BC8
EIGHT	0006	0237	PAPA	006E	0FD2	OFF	0088	13C5	INCH	00A2	17AE
NINE	0010	0395	QUEBEC	00BC	1B7D	ON	00D6	1FCE	INSPECTOR	00F0	23B9
TEN	001A	04E5	ROMEO	010A	2814	IS	0124	2C28	INTRUDER	013E	300A
ELEVEN	0024	0650	SIERRA	0156	33C6	NUMBER	0170	3820	LEFT	018A	3C22
TWELVE	0008	026A	TANGO	0070	1014	TIME	008A	13F1	LOW	00A4	17F4
THIR-	0012	03ED	UNIFORM	00BE	1BC4	CONTROL	00D8	200A	MANUAL	00F2	242A
FIF-	001C	051B	VICTOR	010C	2874	ALERT	0126	2C5F	MEASURE	0140	307D
-TEEN	0026	06B8	WHISKEY	0158	341D	OUT	0172	3880	MILL	018C	3C61
TWENTY	000A	02B3	X-RAY	0072	106C	AUTOMATIC	008C	142E	MOTOR	00A6	1824
HUNDRED	0014	0425	YANKEE	00C0	1C42	ELECTRICIAN	00DA	2071	MOVE	00F4	24A1
THOUSAND	001E	0541	ZULU	010E	28CB	ADJUST	0128	2CBA	NORTH	0142	30CE
A	004E	0B75	AND	015A	3476	POINT	0174	38B5	OF	018E	3C9C
B	0028	06EE	THE	0074	10D0	WAIT	008E	14AC	OPEN	00A8	1869
C	0036	0899	AMPS	00C2	1CA5	AT	00DC	20E2	OVER	00F6	24FC
D	0042	0A1E	HERTZ	0110	2914	BETWEEN	012A	2D1B	PASS	0144	3121
E	0050	0BA3	FARAD 1	015C	34CD	BREAK	0176	3904	PASSED	0190	3CC5
F	002A	0720	WATTS	0076	10F7	SMOKE	0090	14FB	PERCENT	00AA	18B2
G	0038	08DB	MEGA	00C4	1CE0	RED	00DE	2111	PLUS	00F8	2547
H	0044	0A52	MICRO	0112	2948	MINUTES	012C	2D77	POSITION	0146	3172
I	0052	0BD9	MILLI	015E	351D	HOURS	0178	3948	PRESS	0192	3D11
J	002C	075A	METER	0078	1146	ABORT	0092	153C	PROBE	00AC	1904
K	003A	0919	PICO	00C6	1D2A	ALL	00E0	2160	PULL	00FA	2583
L	0046	0A8D	OHMS	0114	299C	BUTTON	012E	2DBD	PUSH	0148	31D1
M	0054	0C1B	CAUTION	0160	3556	CALIBRATE	017A	39AD	RANGE	0194	3D54
N	002E	079D	DANGER	007A	1191	CALL	0094	15A6	READY	00AE	194E
O	003C	095A	FIRE	00C8	1D80	CANCEL	00E2	2197	REPEAT	00FC	258D
P	0048	0ACC	AREA	0116	29F0	CLOCK	0130	2E0B	RIGHT	014A	3201
Q	0056	0C5A	LIGHT	0162	35B1	CRANE	017C	3A21	SAFE	0196	3DB2
R	0030	07DD	PRESSURE	007C	11E4	CYCLE	0096	15E2	SET	0080	198E
S	003E	0988	POWER	00CA	1DD3	DAYS	00E4	21DD	SHUT	00FE	2621
T	004A	0B02	CIRCUIT	0118	2A43	DEVICE	0132	2E49	SLOW	014C	324A
U	0058	0C8E	CHECK	0164	3601	DIRECTION	017E	3A75	SOUTH	0198	3DF2
V	0032	0809	CHANGE	007E	1228	DISPLAY	0098	1630	SPEED	00B2	19CC
W	0040	0988	COMPLETE	00CC	1E21	DOOR	00E6	221F	TEST	0100	2658
X	004C	0B3C	CONNECT	011A	2A9E	EAST	0134	2E97	TOOL	014E	3292
Y	005A	0CCC	DEGREES	0166	3635	ENTER	0180	3AE3	TURN	019A	3E48
Z	0034	085D	MINUS	0080	1291	EQUAL	009A	168B	UNDER	00B4	1A16
ALPHA	005C	0D0E	REPAIR	00CE	1E8B	EXIT	00E8	226C	VOLTS	0102	2696
BRAVO	0062	0DE4	SECONDS	011C	2AEB	FAIL	0136	2ED8	WEST	0150	32D1
CHARLIE	0066	0EB0	SERVICE	0168	36CB	FEET	0182	3B17	YELLOW	019C	3E9B
DELTA	0060	0D91	NOT	0082	12E1	FAST	009C	16DD	OPERATOR	00B6	1A6A
ECHO	005E	0D4F	TEMPERATURE	00D0	1EEC	FLOW	00EA	228A	GALLONS	0104	26D3
FOXTROT	0064	0E41	UNIT	011E	2B55	FREQUENCY	0138	2F1D			
GOLF	0068	0F01	SWITCH	016A	3721	FROM	0184	3B56			

Table 4. Distribution of frequencies in the case study data.

Available data	7766 frames (both voiced and unvoiced)		
Sample size	5464 frames (voiced)		
Maximum frame frequency	32		
		Percentage of Distinct Frame Types	Percentage of All Frames
# of distinct frames	2855		
# of frequency 1 frames	1717	60.1	31.4
# of frequency 2 frames	566	19.8	20.7
# of frequency 3 frames	245	8.6	13.5
# of frequency 4 frames	145	5.1	10.6
# of frequency 5 frames	58	2.0	5.3
# of frequency 6 frames	52	1.8	5.7
# of frequency 7 frames	17	0.6	2.2
# of frequency 8 frames	22	0.8	3.2
# of frequency 9 frames	8	0.3	1.3
# of frequency $\geq$ 10 frames	331	0.9	6.1
		100	100

4, from the table we see that $t_{.975}$ is 2.78. The confidence interval on μ_1 is $.0512 < \mu_1 < .0528$. The above exercise could not be carried out for the case study, since only one data set was available. Therefore, the frequency number for the frames are used to compute the probabilities of the letters for a variety of different λ values, regardless of their confidence levels.

Average Code Length

With the assumption that probabilities are known, this study proceeds to calculate the upper bound on the average code length.

The average code length, l_{ave}, is bounded by

$$l_{ave} < \sum_{m=1}^{M} q_m \times \log_2(1/q_m) + l_s \left(1 - \sum_{m=1}^{M} q_m\right) + 1$$

and is dependent on the value of parameters c and λ. In the above inequality, l_s can be derived readily. For each c value, $l_{s1} = \log_2(T/c) + 40$ and $l_{s2} = \log_2[1/(1 - \sum_{m=1}^{M} q_m)] + 39$ (refer to proposed solution). Figures 5 through 8 show the plot of l_{ave} vs. c with λ as an independent parameter. In Figure 5, $q_i = (t_i + \lambda)/(T + N \times \lambda)$ if $t_i/T \geq c$ (case I), and in Figure 6, $q_i = (t_i + \lambda)/(T + N \times \lambda)$ if $q_i \geq c$ (case II), $l_s = l_{s1}$ for Figures 5 and 6. In Figures 7 and 8, $q_i = (t_i + \lambda)/(T + N \times \lambda)$ if $t_i/T \geq c$ and $q_i = (t_i + \lambda)/(T + N \times \lambda)$ if $q_i \geq c$, respectively. For both these figures $l_s = l_{s2}$.

From Figure 5 it can be concluded that for $\lambda > 2^{-26}$, l_{ave} will always be greater than 39 bits (for $l_s = l_{s1}$), thus making the application of variable code length technique unattractive. For $\lambda < 2^{-26}$ and $c < 2/T$, l_{ave} will become smaller than 39 bits (for $l_s =$

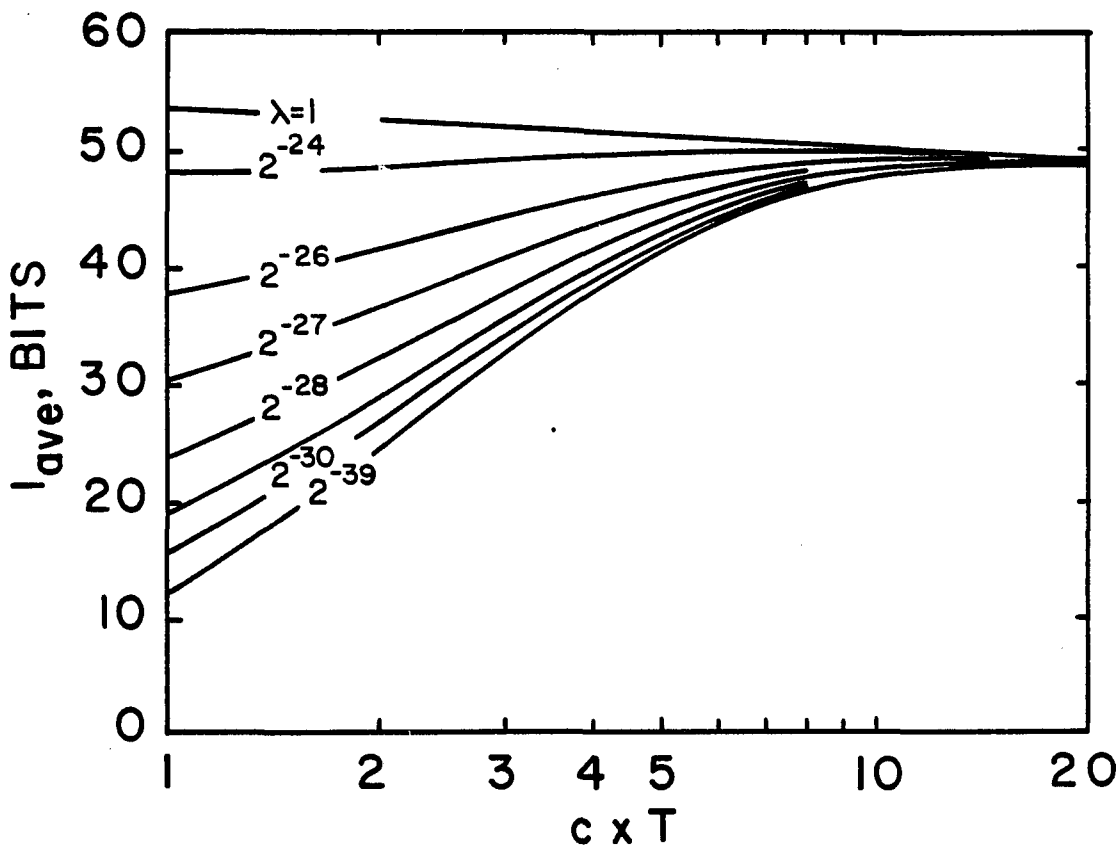


Figure 5. Average code length, l_{ave}, as a function of c and λ, case I, $l_s = l_{s1}$.

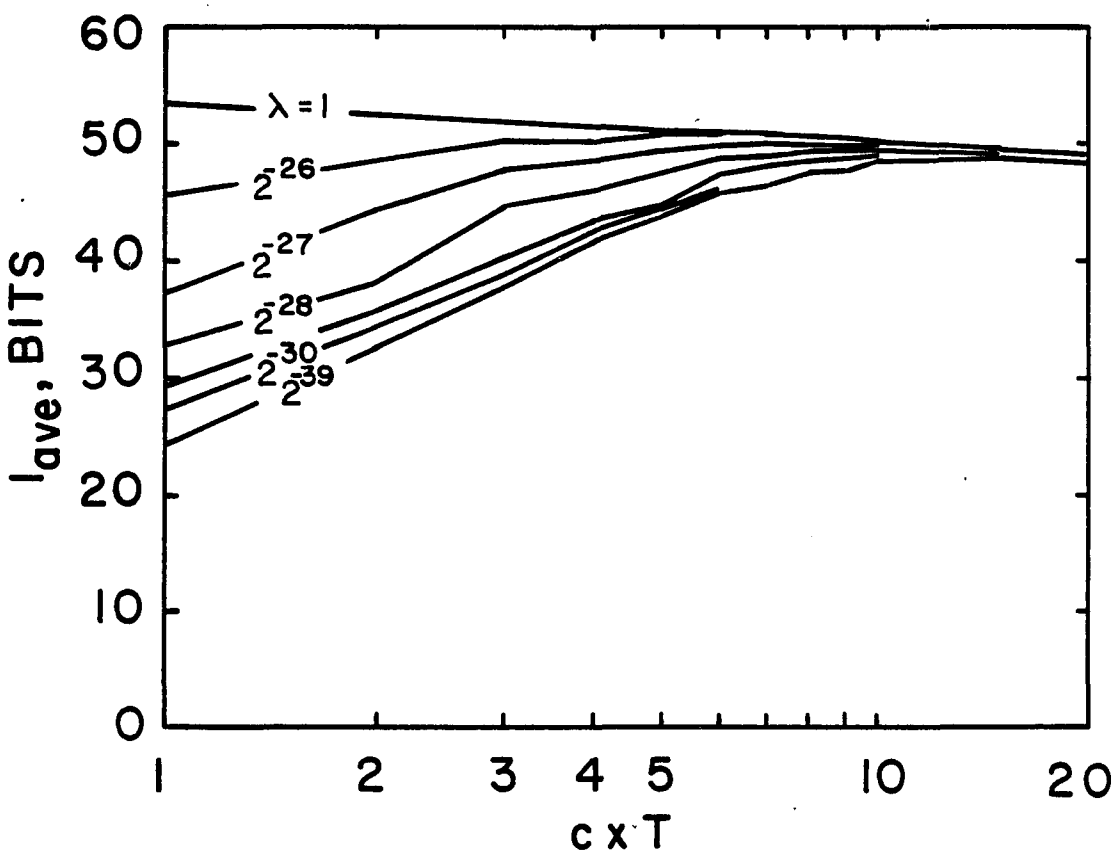


Figure 6. Average code length, l_{ave}, as a function of λ and c, case II, $l_s = l_{s1}$.

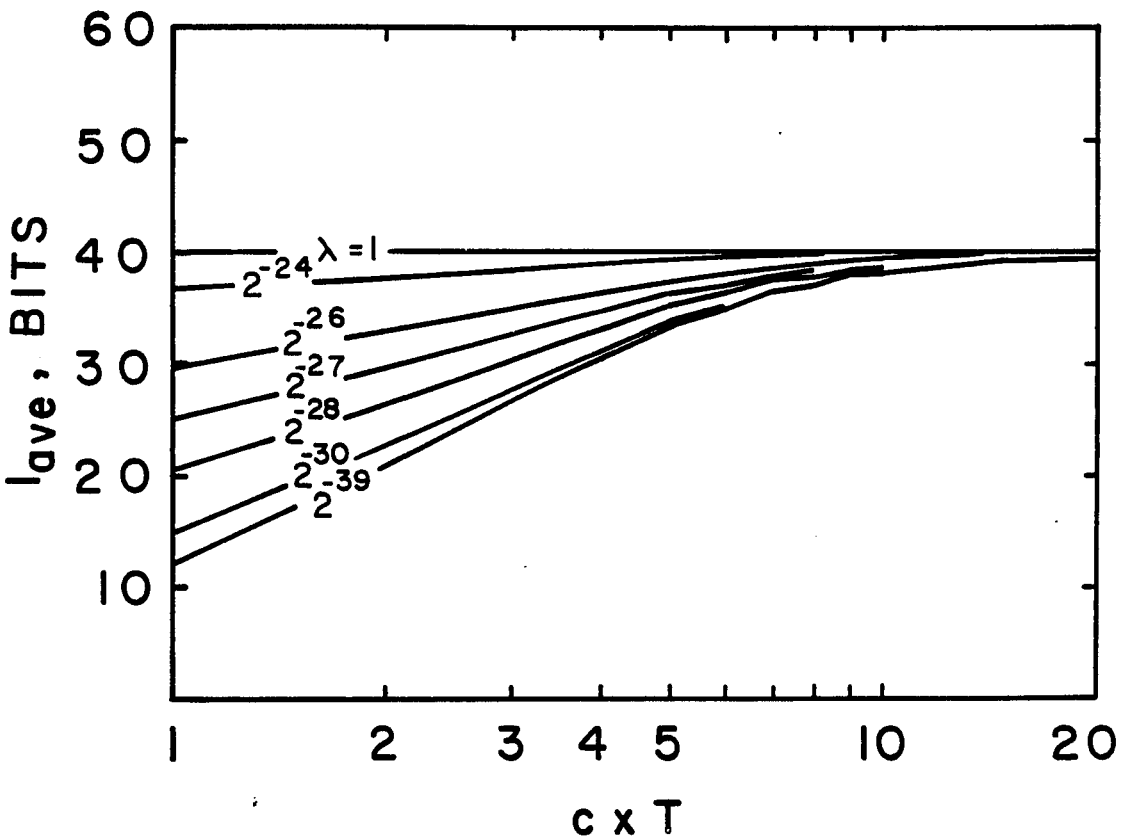


Figure 7. Average code length, l_{ave}, as a function of λ and c, case I, $l_s = l_{s2}$.

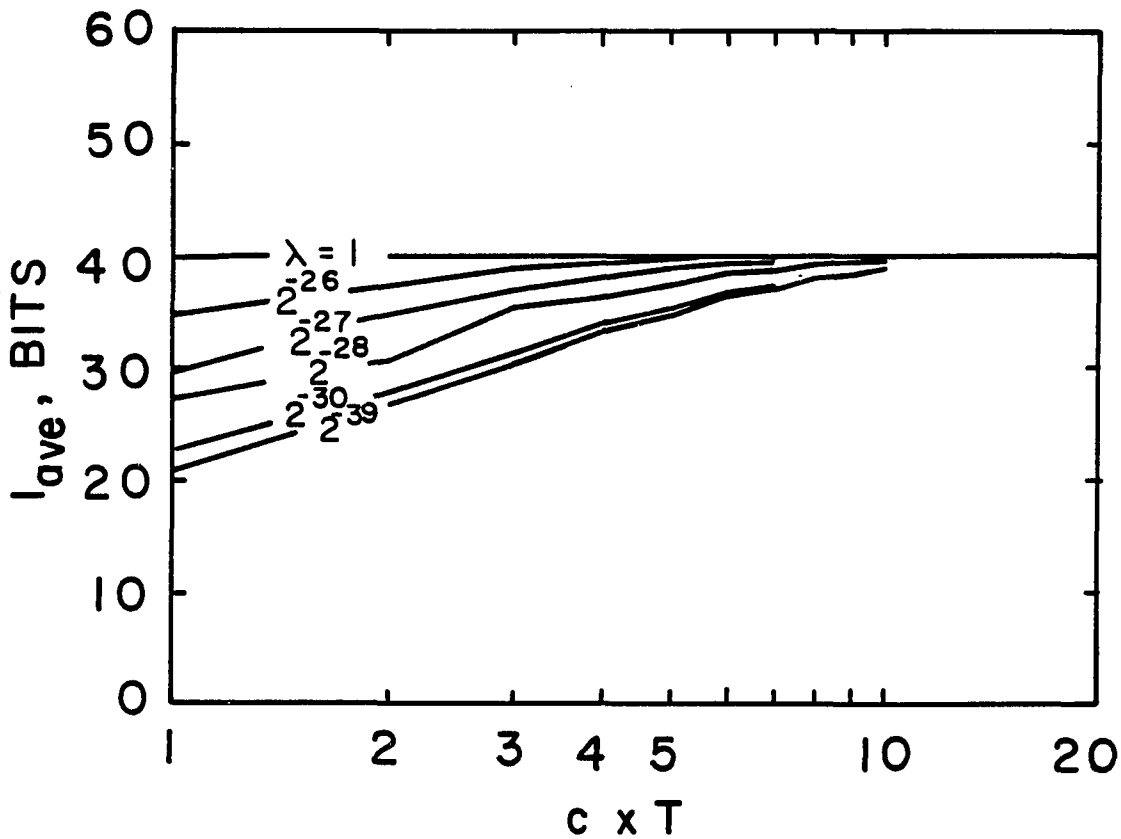


Figure 8. Average code length, l_{ave}, as a function of λ and c, case II, $l_s = l_{s2}$.

l_{s1}). For smaller λ's (e.g., $\lambda = 2^{-39}$) l_{ave} becomes considerably smaller than 39 bits.

Figure 6 shows the same general behavior as Figure 5, except that the number of q_i's $>$ c differs from those of Figure 5.

It is also apparent from both figures that for $c \geq 10/T$, λ is not an important factor and l_{ave} approaches the limiting value of 49 bits (for $l_s = l_{s1}$). In other words, when the cutoff probability value c is greater than 10/T, different values of parameter λ do not change the average code length significantly.

In Figures 7 and 8, the limiting value of l_{ave} is 40 bits. As was the case in Figures 5 and 6, the difference between Case I and Case II is that the number of q_i's $\geq$ c differs for the two figures. For these two figures, the variable code length technique will always result in smaller l_{ave}, for all values of c and λ. This is so because l_{s2} is a function of sample size whereas l_{s1} is only a function of cutoff value c.

In general, l_{s2} is a better (tighter) upper bound for the code length which in turn results in a more efficient coding of the case study data.

Storage Requirement

The average storage requirements corresponding to both Figures 5 and 6 are presented in Figure 9. This storage requirement is calculated by l_{ave} x T. Figure 10 shows the average storage requirement corresponding to l_{s2} (Figures 7 and 8).

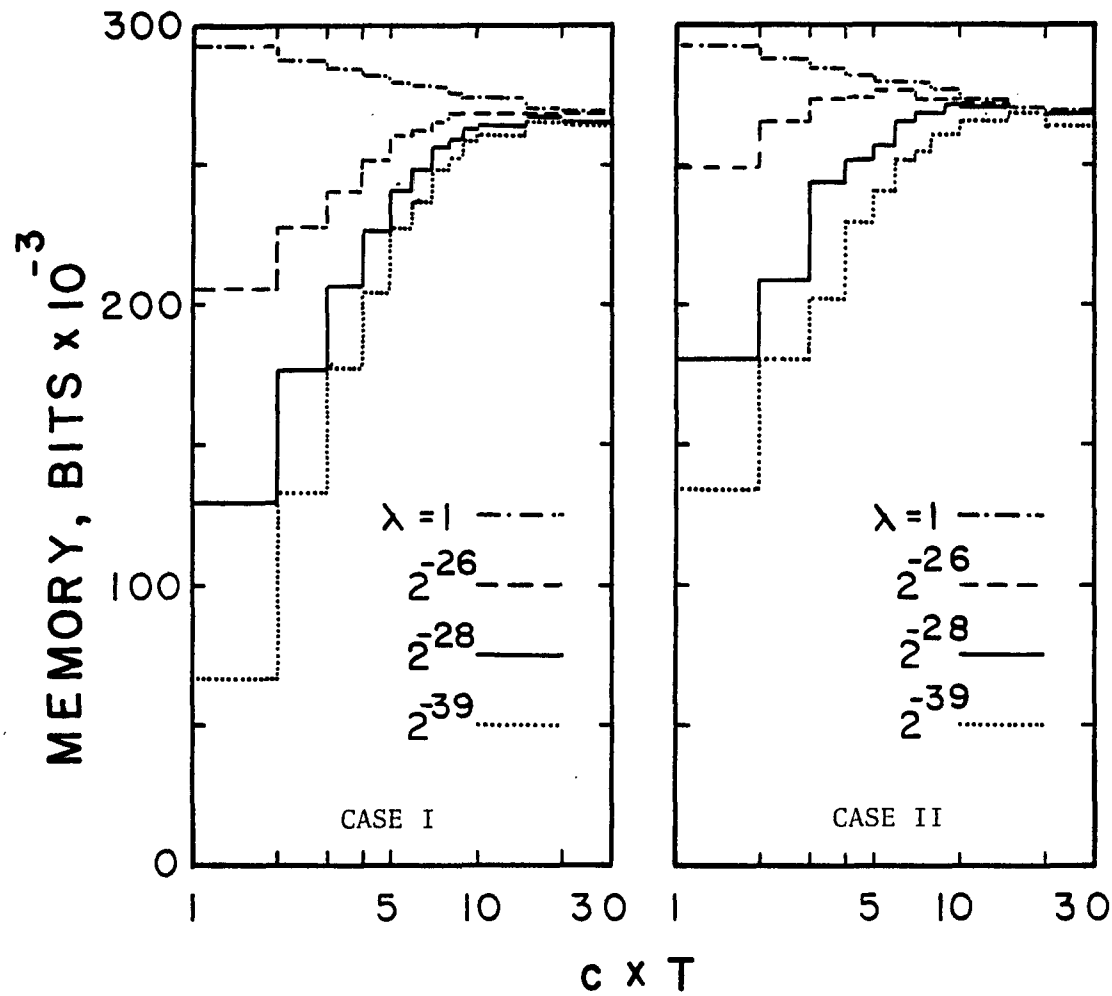


Figure 9. Average storage requirements as a function of λ and c. -- $l_s = l_{s1}$.

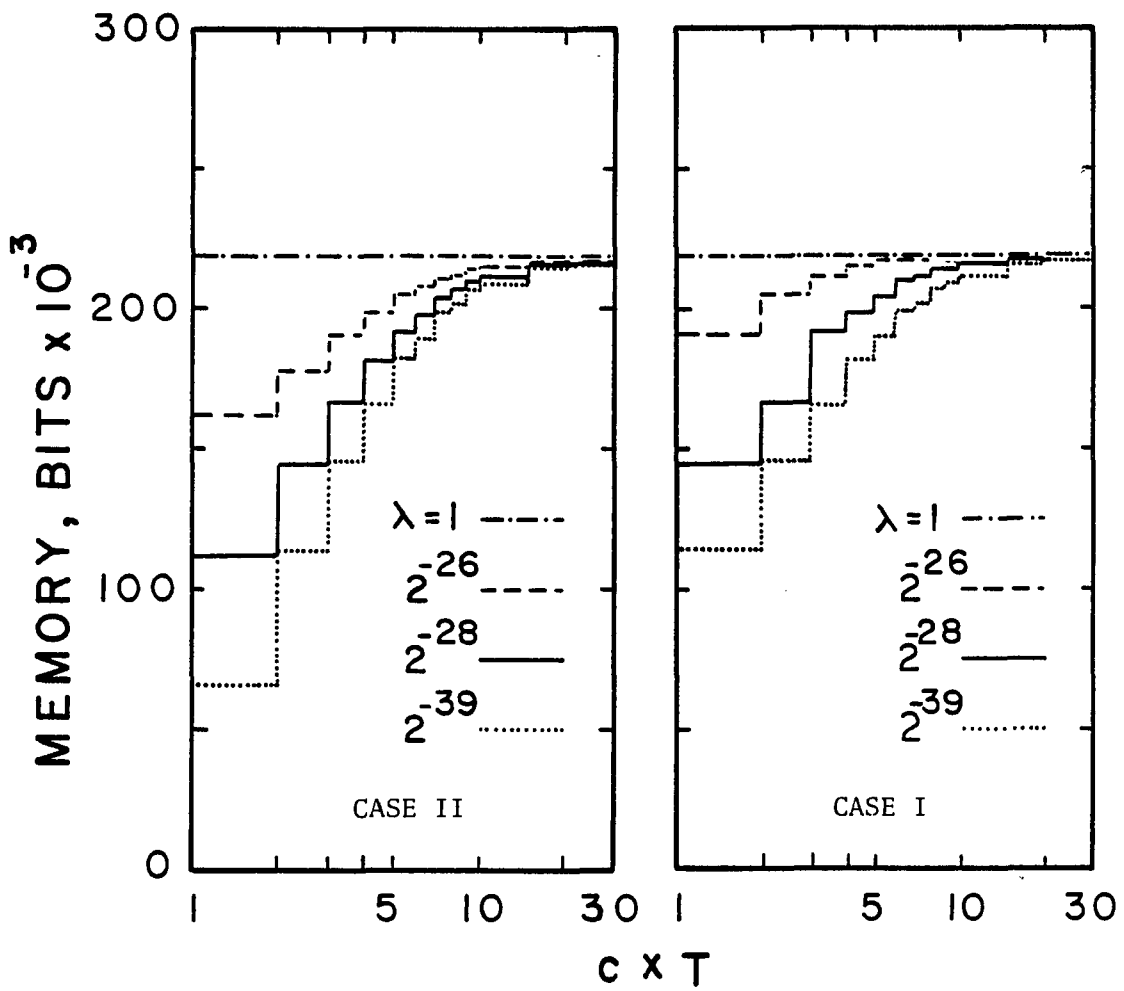


Figure 10. Average storage requirements as a function of λ and c. -- $l_s = l_{s2}$.

Using the Shannon-Fano coding technique (i.e. $l_i < \log_2(1/q_i) + 1$), the 5464 data frames in this study, are encoded and the total storage requirement is calculated for different and c values. Figures 11 through 14 show the total storage in bits as a function of c and λ. The programs used for these calculations are LCODE and LCODEH. For a fixed code length of 39 bits, the storage requirement is 213 K bits. The variable length code technique could result in using one-third of the storage requirement (63.38 K) for fixed code, which is a significant saving in storage memory. Table 5 summarizes the storage requirement calculations for $\lambda = 2^{-39}$, $c = 1/T$, and $l_s = l_{sl}$.

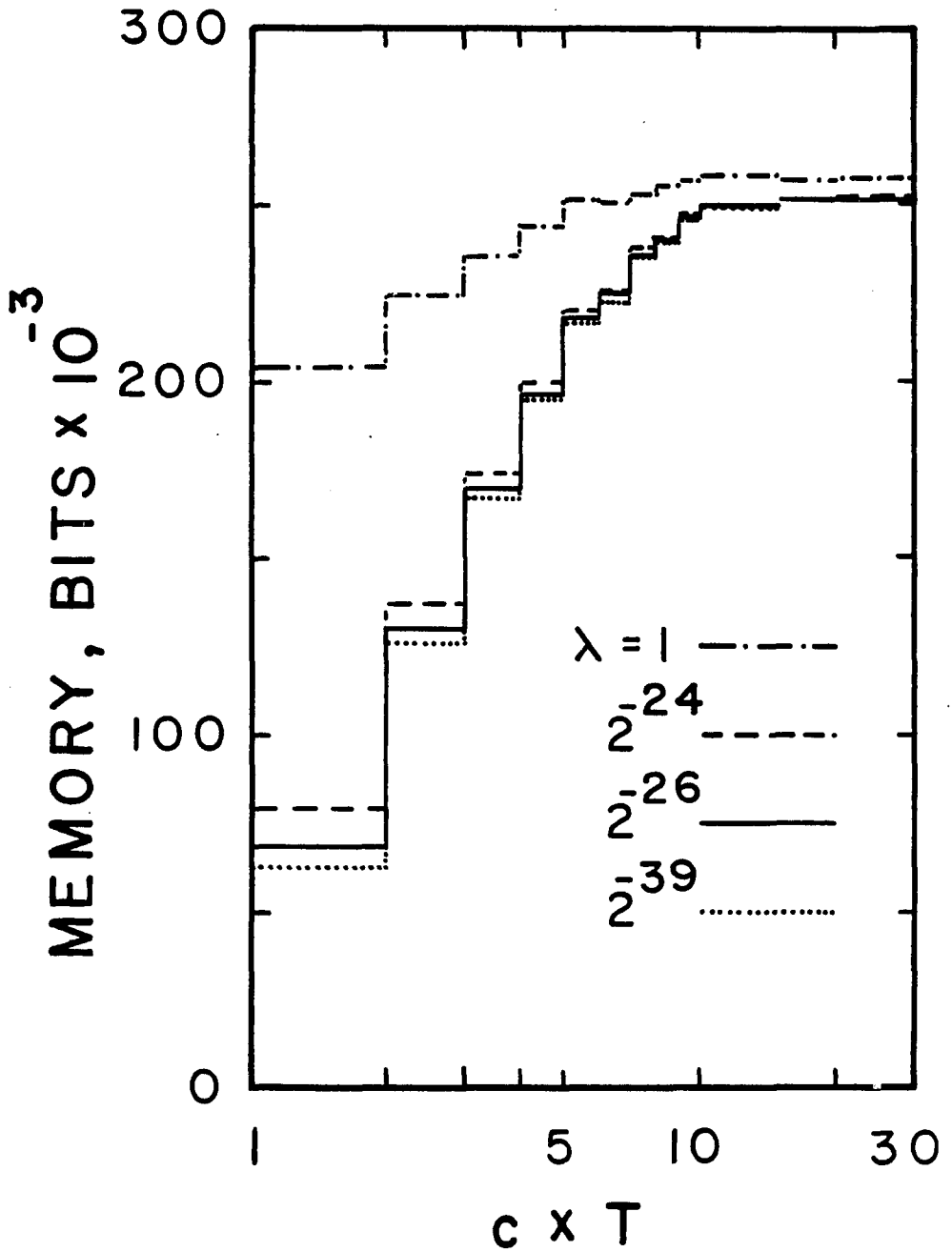


Figure 11. Total storage requirements using the Shannon-Fano coding, case I, $l_s = l_{s1}$.

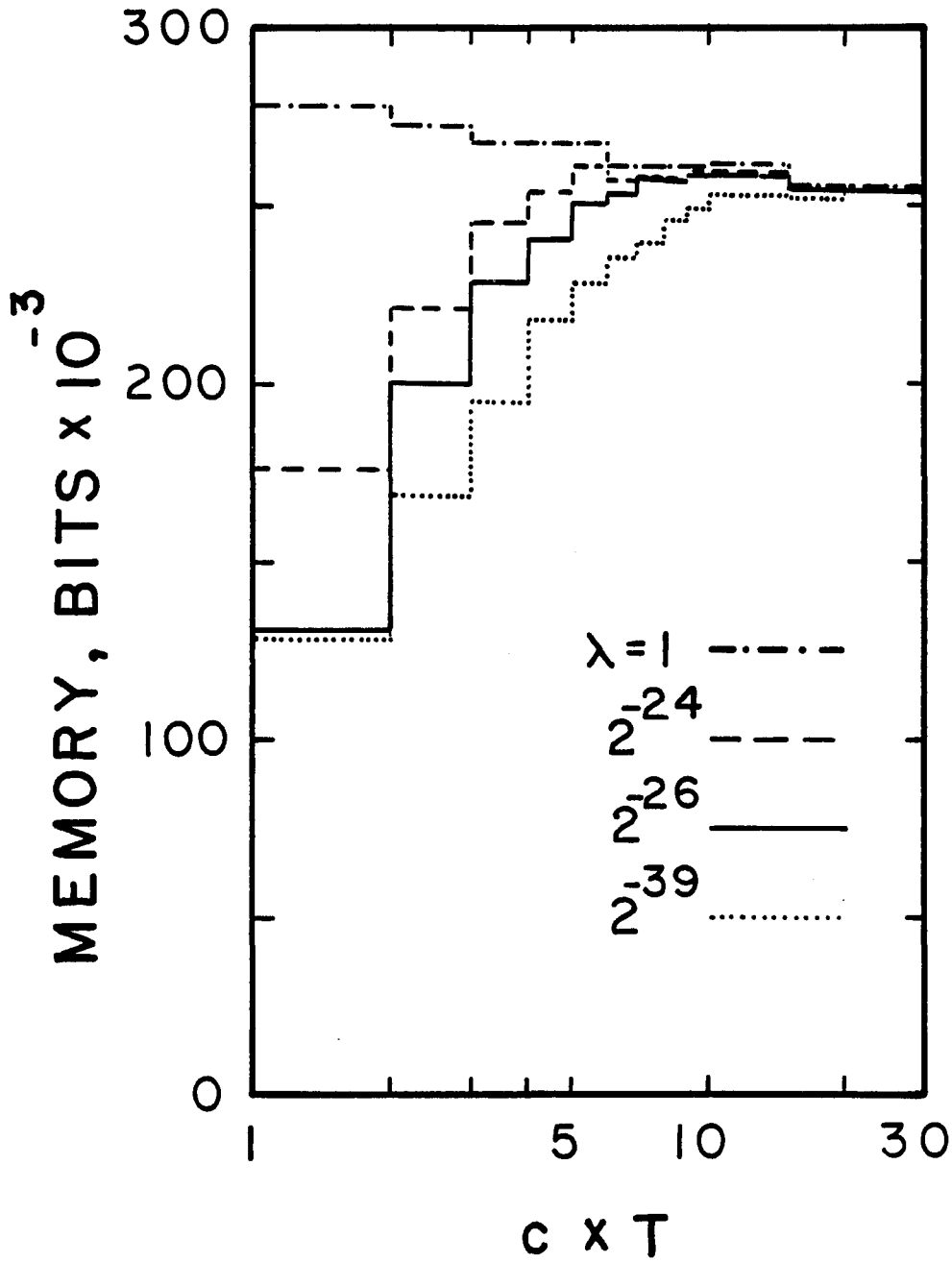


Figure 12. Total storage requirements using the Shannon-Fano coding, case II, $l_s = l_{s1}$.

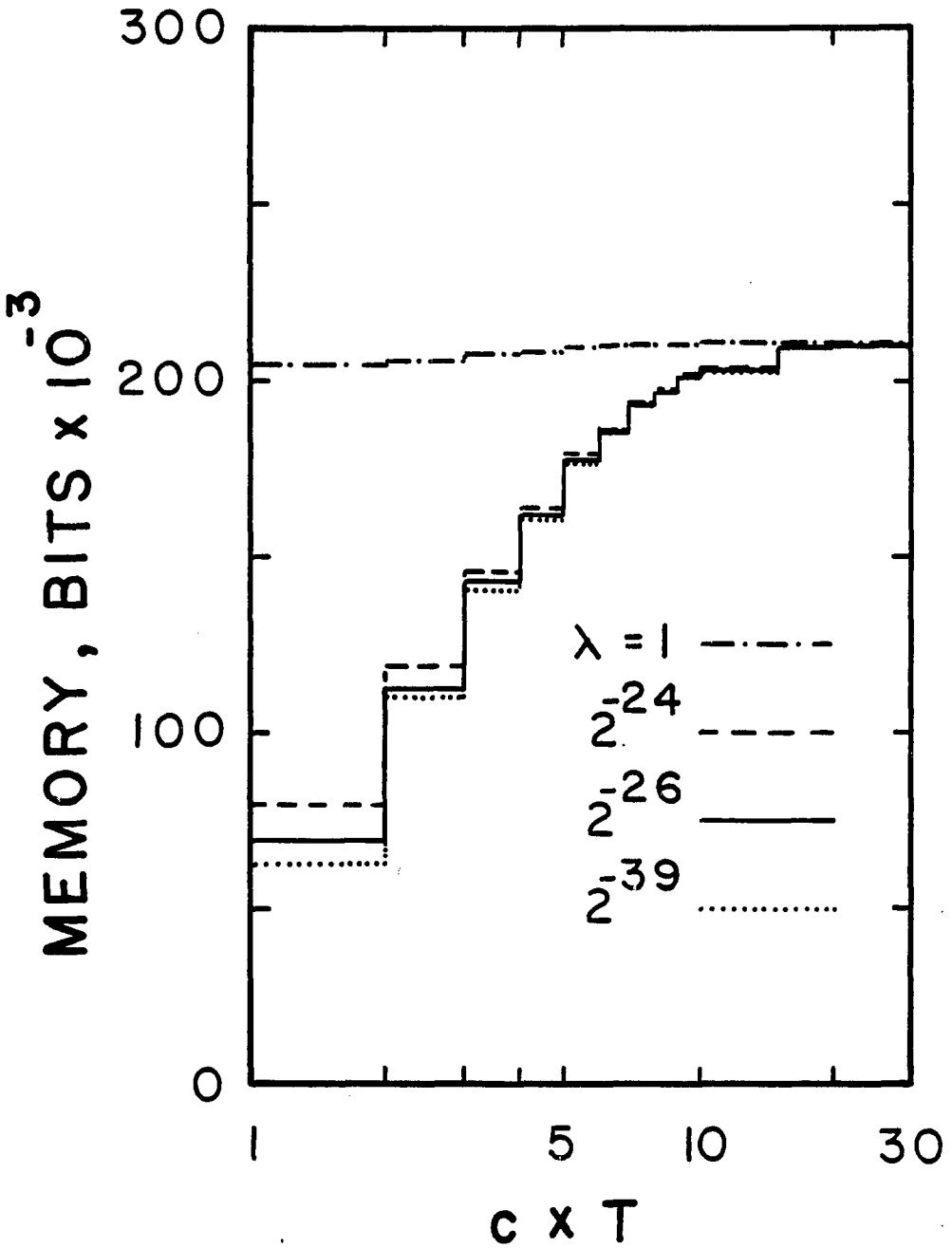


Figure 13. Total storage requirements using Shannon-Fano coding, case I, $l_s = l_{s2}$.

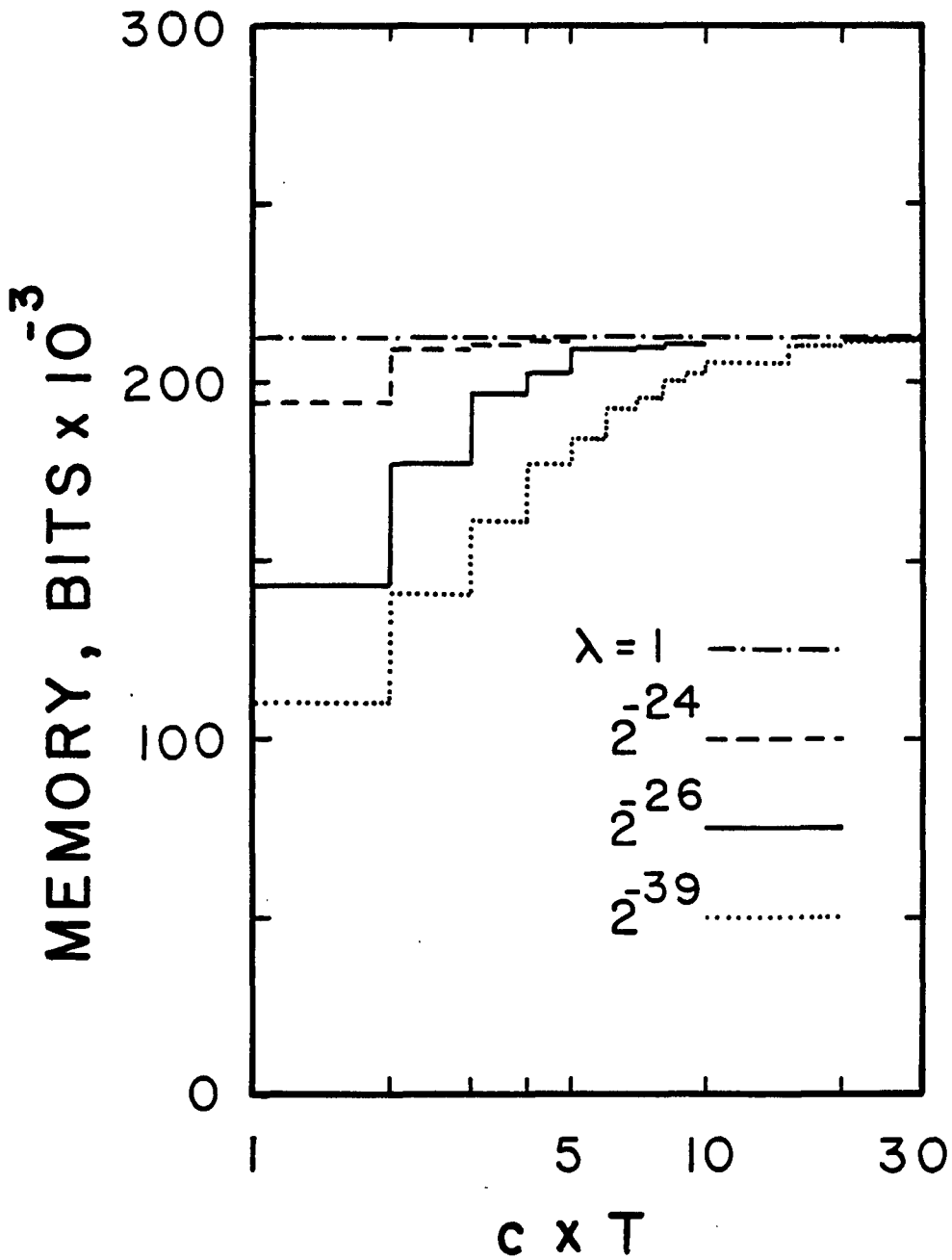


Figure 14. Total storage requirements using the Shannon-Fano coding, case II, $l_2 = l_{s2}$.

Table 5. Calculation of total storage memory for the case study data, using the Shannon-Fano coding.

Frequency of Obsevation	q	No. of Occurrence of Same Frequency	Code Length	Total Code Length
1	1/5464	1717	13	22321
2	2/5464	566	12	3584
3	3/5464	245	11	8085
4	4/5464	145	11	6380
5	5/5464	58	11	3190
6	6/5464	52	10	3120
7	7/5464	17	10	1190
8	8/5464	22	10	1760
9	9/5464	8	10	720
10	10/5464	11	10	1100
11	11/5464	3	9	297
12	12/5464	5	9	540
14	14/5464	1	9	126
15	15/5464	1	9	135
20	20/5464	2	9	360
27	27/5464	1	8	216
32	32/54	1	8	256
Total Storage Memory				63380

SUMMARY AND CONCLUSIONS

Presently the TMS 5220 accepts input data frames that are a string of 39 binary digits. To store data frames, T x 39 bits of memory are required (T is the sample size). In the proposed solution the steps to be taken for employment of a variable code length technique are studied. The Shannon-Fano coding is chosen as an upper bound on the average code length. This upper bound certainly bounds the more efficient coding algorithms, specifically the Huffman coding. Computer programs to calculate the probabilities, code lengths, and the average code length are developed.

An upper bound to the average code length is suggested. This upper bound depends on the values of probability estimates and a probability cutoff value of c. Also, in calculating the probabilities, a method of biasing the empirical probabilities has been investigated. This biasing acts as an insurance against grossly inefficient code lengths in case of an uncommon input sample data. Two approaches for finding an upper bound for code length of non-frequent, l_s, events are investigated. $l_s = l_{s2} = \log_2[1/(1 - \sum_{m=1}^{M} q_m)] + 39$ is a tighter bound and results in a shorter average code length in this study. It is conjectured that this would be true in general.

The problems associated with obtaining statistical confidence levels for the probability estimates of the commmon letters are discussed. The main obstacle in pursuing the proposed steps, for calculating confidence levels on the probabilities, was insufficient speech data sets.

In the case study section, a sample data consisting of 5464 voiced data frames is analyzed according to the proposed methods. Plots of the average code length vs. cutoff value c and biasing parameter λ are presented. Several plots explain the dependency of storage requirements on λ and c.

The results of the case study suggest that, for appropriate λ and c, a variable code length coding appears superior to the fixed code length. For the specific data analyzed, the required storage is one-third of the fixed code length requirements.

For further studies in this area the author suggests the following:

1. Acquiring different sources of English speech data to study the sensitivity of the proposed method to the differences of speech sources.
2. A confidence intervals on the estimate of average code length can be found after the probability values for all the alphabet letters have been estimated and used in l_{ave} derivation. The implementation of this approach is beyond the scope of this study due to the mathematical complexity of deriving a distribution for l_{ave} [8], and insufficient samples of speech data. A

good reference for a similar statistical approach can be found in Baggeroer's work [2].

3. In the case of a sample data with more than 8000 voiced data frames, either one of the following approaches are suggested. First, the data can be divided into sections small enough to be inputted to the developed program, and each division can be handled separately. Second, another sorting program can be written to input the resulting sorted outputs of the present program. Then sort these inputs as a whole. The sorting comparisons for this approach are not large since the input is already partially sorted [11].

Finally, this study shows that implementation of a variable length coding technique, for storing speech data for the TMS 5220, could result in storage memory savings.

APPENDIX

LISTINGS OF COMPUTER PROGRAMS

Listing of FREQAS program.

```
program freqas(input,output,infile,ofile,o2file,o3file);

(* this program performs a sorting on the input data ,then strips  *)
(* the adjacent duplicate data frames and stores a frequency count *)
(* for each distinct frame.                                        *)
(* it also eliminates the frames for which either pitch or energy  *)
(* values are zero.                                                *)
(* procedures called by this program are:                          *)
(*      ._getasm                                                   *)
(*      ._putasm                                                   *)
(*      ._quicksort                                                *)
(*      ._unique                                                   *)
(*      ._entropy                                                  *)
(* the input to this programs comes from infile which in turn is   *)
(* sent to the main memory via putasm procedure.                   *)
(* the output of the main program is a message on display in case  *)
(* of no input data.                                               *)
(* other outputs are genrated from the procedures.                 *)
(*                                                                 *)
(*                                                                 *)
(*                                                                 *)

const    abovelts= 501;                 (*bouding constant *)
         numelts = 500;                 (*bouding constant *)
         maxstack = 500;                (*max cap. of stack array*)

type     pararr    = array[1..10] of integer; (*buffer to store a frame *)
         aptr      = 1..numelts;
         aptr2     = 0..abovelts;


var      frame   : pararr;               (*input unit*)
         k       : pararr;
         a,b     : pararr;
         i,j,l   : aptr;
         und     : boolean;
         aneb    : boolean;
         ageb    : boolean;
         aleb    : boolean;
         energy  : integer;
         pitch   : integer;
         flag    : integer;
         infile  : text;                (*input file*)
         ofile   : text;                (*outputfile for quicksort*)
         o2file  : text;                (*outputfile for unique*)
         o3file  : text;                (*outputfile for input*)


procedure          putasm(var k:integer; i:integer);       forward;
procedure          getasm(var k:integer; i:integer);       forward;


procedure entropy;

(* this pocedure calculates the total entropy associated with the *)
(* data from o2file.                                              *)
(* the input comes from o2file.                                   *)
(* the output is the value of entropy on the display.             *)

var      n     : xreal;
         total : xreal;
         H     : xreal;
         log2e : xreal;
```

Listing of FREQAS program.

```
begin   (*entropy*)
  log2e := 1.0/ln(2.0);
  reset(o2file);
  total := 0.0;
  H := 0.0;
  while not eof(o2file) do
                          begin
                          readln(o2file,n);
                          total := total + n;
                          end (*while not eof*);
  reset(o2file);
  while not eof(o2file) do
                          begin
                          readln(o2file,n);
                          H := H + (n/total) * log2e * ln(total/n)
                          end  (*while not eof*) ;
  writeln('entropy =  ',H)
  end;  (* entropy*)


procedure unique;

(* this procedure strips adjacent duplicate lines.                    *)
(* the input file to this routine is the outputfile of quicksort*)
(* ,ofile.                                                            *)
(* the output file is o2file                                          *)
(*                                                                    *)
(*                                                                    *)
type     Karr  = array[1..10] of integer;

var      buf1,buf2 : Karr;
         t         : Karr;
         n         : integer;
         i         : integer;

function aneb(var a: pararr; var b:pararr):boolean;
(* thif function tests two frames for inequality                      *)
(*                                                                    *)
var      index: integer;

begin (*aneb*)
  index:=1;
  repeat
        if a[index] = b[index] then index :=index  +1
  until (a[index] () b[index]) or (index = 11);
  if index = 11 then aneb:= false
  else   aneb := true
end;   (*aneb*)


begin  (*unique *)
  rewrite(o2file);
  reset(ofile);
  if eof(ofile) then writeln('input is empty for unique')

  else begin

  readln(ofile,t[1],t[2],t[3],t[4],t[5],t[6],t[7],t[8],t[9],t[10]);
  for i:=1 to 10 do   buf1[i]:= t[i];
  if eof(ofile) then begin
       n := 1;
       writeln(o2file,n,buf1[1]:4,buf1[2]:4,buf1[3]:4,buf1[4]:4,
```

Listing of FREQAS program.

```
                    buf1[5]:4,buf1[6]:4,buf1[7]:4,buf1[8]:4,
                    buf1[9]:4,buf1[10]:4)
                end  (*then begin *)
else begin

  while   not eof(ofile)   do
     begin
     n := 0;
     repeat
          readln(ofile,t[1],t[2],t[3],t[4],t[5],t[6],
                                  t[7],t[8],t[9],t[10]);
          for i:=1 to 10 do  buf2[i]:= t[i];
          n := n+1
     until (aneb(buf1,buf2)) or (eof(ofile));
     if (eof(ofile) and (not(aneb(buf1,buf2))))   then n:=n+1;
     writeln(o2file,n,buf1[1]:4,buf1[2]:4,buf1[3]:4,buf1[4]:4,buf1[5]:4,
                 buf1[6]:4,buf1[7]:4,buf1[8]:4,buf1[9]:4,buf1[10]:4);
     if (eof(ofile) and aneb(buf1,buf2)) then
              begin
              n := 1;
              writeln(o2file,n,buf2[1]:4,buf2[2]:4,buf2[3]:4,
                              buf2[4]:4,buf2[5]:4,buf2[6]:4,
                              buf2[7]:4,buf2[8]:4,buf2[9]:4,
                              buf2[10]:4)
              end; (*then begin*)


                if not eof(ofile) then
                                  begin
                                  n :=0;
                                  repeat
                                    readln(ofile,t[1],t[2],t[3],t[4],
                                                t[5],t[6],t[7],t[8],
                                                t[9],t[10]);
                                    for i:=1 to 10 do buf1[i]:=t[i];
                                    n := n+1
                                  until (aneb(buf1,buf2)) or (eof(ofile));
             if (eof(ofile) and (not(aneb(buf1,buf2)))) then n:=n+1;
              writeln(o2file,n,buf2[1]:4,buf2[2]:4,buf2[3]:4,
                              buf2[4]:4,buf2[5]:4,buf2[6]:4,buf2[7]:4,
                              buf2[8]:4,buf2[9]:4,buf2[10]:4);


              if (eof(ofile) and aneb(buf1,buf2)) then begin
                              n := 1;
                              writeln(o2file,n,buf1[1]:4,
                                      buf1[2]:4,buf1[3]:4,buf1[4]:4,
                                      buf1[5]:4,buf1[6]:4,buf1[7]:4,
                                      buf1[8]:4,buf1[9]:4,buf1[10]:4)
                                               end (*then begin*)


                                  end  (*then begin*)
             end  (* while do begin*)
  end   (*else begin*)
  end   (*else begin*)
end;  (*unique*)



procedure quicksort( n:aptr);
```

Listing of FREQAS program.

```
(*this procedure sorts the input data stored in the main memory in   *)
(*the descending order and puts each datum on a line.                *)
(* the output of this routine is the sorted data in the main memory. *)
(*                                                                    *)
(*                                                                    *)

type stackitem = record
                   lb : aptr2;
                   ub : aptr2;
                 end;

     stack = record
               top : 0..numelts;
               item : array[1..numelts] of stackitem
             end;

var s : stack;
    newbnds : stackitem;
    i,j : aptr;

function ageb(var a:pararr; var b:pararr):boolean;
(*this function compares frame a with frame b and gets set if a)=b*)
var      index : integer;

begin
index :=0;
repeat
        index :=index +1
until (a[index] <> b[index]) or (index > 10);
if index = 11 then ageb:= true
else
        begin
        if a[index] < b[index]  then ageb:= false
        else      ageb:=true
        end (*else begin*)
end;  (*function ageb*)


function aleb(var a:pararr; var b:pararr):boolean;
(*this function compares frame a with frame b and gets set if a(=b*)
var      index : integer;

begin
index :=0;
repeat
        index :=index +1
until (a[index] <> b[index]) or (index > 10);
if index = 11 then aleb:= true
else
        begin
        if a[index] > b[index]  then aleb:= false
        else      aleb:=true
        end (*else begin*)
end;  (*function aleb*)

function empty(s:stack):boolean;
(*this function checks the stack to see if it is empty        *)
begin
  if s.top = 0  then empty := true
  else               empty := false
end;   (*function empty*)
```

Listing of FREQAS program.

```
procedure pop(var s:stack; var x:stackitem; var und:boolean);
(*this procedures takes a member off the stack                *)
begin
  if empty(s) then und := true
  else   begin
         und := false;
         x := s.item[s.top];
         s.top := s.top -1
         end  (*else begin*)
end;   (* pop*)

procedure push(var s:stack; var x:stackitem; var und:boolean);
(* this procedure puts a datum on thew stack                  *)
begin
  if s.top = maxstack then und := true
  else   begin
         s.top := s.top +1;
         s.item[s.top] := x
         end  (*else begin*)
end;    (*push*)


procedure rearrange(lb,ub: aptr2; var j: aptr);

(*this procedure rearranges the input array about a pivot point so   *)
(*that all the elements to one side of the pivot are either larger   *)
(*or smaller than the pivot point.                                   *)

var up,down: aptr;
    a      : pararr;
    i      : integer;

begin   (*rearrange*)
  getasm(frame[1],lb);
  for i:= 1 to 10 do  a[i]:= frame[i];
  j:= lb;
  up:= ub;
  down:= lb;
  getasm(frame[1],up);
  repeat
        while(up ) down) and (ageb(frame,a))
                        do begin
                        up:= up-1;
                        getasm(frame[1],up)
                        end; (*while do begin*)
        j:=up;
        if up() down
            then  begin
                  getasm(frame[1],up);          (*frame[down] = frame[up]*)
                  putasm(frame[1],down);
                  getasm(frame[1],down);
                  while(down ( up) and (aleb(frame,a))
                                  do begin
                                  down:= down +1 ;
                                  getasm(frame[1],down)
                                  end  (*while do begin*);
                  j:= down;
                  if down()up
                           then begin
                                getasm(frame[1],down);
                                putasm(frame[1],up)
                                end  (*if then begin *)
```

Listing of FREQAS program.

```
                end   (*then begin*)
  until down = up;
  putasm(a[1],j)
end;  (*rearrange*)


begin   (*quicksort*)
  stop := 0;
  with newbnds
       do       begin
                lb:= 1;
                ub:= n;
                push(s,newbnds,und);
                while not empty(s)
                     do  begin
                         pop(s,newbnds,und);
                         while  ub > lb
                              do  begin

                                  rearrange(lb,ub,j);
                                  if j-lb > ub-j
                                      then  begin
                                            i:= ub;
                                            ub:= j-1;
                                            push(s,newbnds,und) ;
                                            lb:= j+1;
                                            ub:=i
                                            end (*then begin*)
                                  else      begin
                                            i:=lb;
                                            lb:= j+1;
                                            push(s,newbnds,und);
                                            lb:=i;
                                            ub:=j-1;
                                            end  (*else begin*)
                         end   (*while ub>lb do begin*)
                end   (*while not empty(s) do begin*)
       end   (*with do begin*)
end;  (*quicksort*)


begin   (*freq*)
(*this is the main program.                                    *)

  i      := 1;
  rewrite(ofile);
  reset(infile);
  while not eof(infile)   do   (*read all data put them in frame*)

                begin
                readln(infile,energy,pitch,K[1],K[2],K[3],K[4],K[5],
                                           K[6],K[7],K[8],K[9],K[10]);
                flag := pitch * energy;
                if flag <> 0 then
                         begin
                         putasm(K[1],i);
                         i:= i+1
                         end (* if then begin *)
                end;
  rewrite(o3file);
  j:= i-1;
```

Listing of FREQAS program.

```
    for i:=1 to j do begin
                    getasm(frame[1],i);
                    writeln(o3file,frame[1],frame[2],frame[3],frame[4],
                                frame[5],frame[6],frame[7],frame[8],
                                frame[9],frame[10])
                    end;      (*do begin*)


    if i>1 then     begin
                    i:=i-1;
                    quicksort(i);
                    for i := 1 to i do  begin
                                        getasm(frame[1],i);
                                        writeln(ofile,frame[1],frame[2],
                                          frame[3],frame[4],frame[5],
                                          frame[6],frame[7],frame[8],
                                          frame[9],frame[10])
                                        end  (*do begin*)
                    end  (*then begin*)
    else            begin
                    write('input is empty')
                    end (*else begin*);
    unique ;
    entropy

end.  (*freq*)
```

REFERENCES CITED

1. Ash, R. B., Information Theory, Interscience, New York, 1965.

2. Baggeroer, A. B., Confidence Intervals for Regression (MEM) Spectral Estimates, IEEE Transactions on Information Theory, Vol. IT-22, No. 5, September 1976, pp. 534-545.

3. Campbell, L. L., Kraft Inequality for Decoding with Respect to a Fidelity Criterion, IEEE Transactions on Information Theory, January 1973, pp. 68-73.

4. Crolette, A. and J. Pearl, Bounds on Memory vs. Error Trade-offs in Question Answering Systems, IEEE Transactions on Information Theory, Vol. IT-25, No. 2, March 1979, pp. 193-201.

5. Crolette, A. and J. Pearl, Elasticity Conditions for Storage versus Error Exchange in Question-Answering Systems, IEEE Transactions on Information Theory, Vol. IT-25, No. 6, November 1979, pp. 653-664.

6. Davisson, L., R. J. McEliece, M. B. Pursley and M. S. Wallace, Efficient Universal Noiseless Source Codes, IEEE Transactions on Information Theory, Vol. IT-27, No. 3, May 1981.

7. Dobrushin, R. L., Translated by E. R. Berllkamp, Survey of Soviet Research in Information Theory, IEEE Transactions on Information Theory, Vol. IT-18, No. 6, November 1972, pp. 703-721.

8. Feller, W., An Introduction to Probability Theory and Its Applications, Vol. I, 3rd edition, John Wiley & Sons, Inc., 1968.

9. Gilbert, E. N., Codes Based on Inaccurate Source Probabilities, IEEE Transactions on Information Theory, Vol. IT-17, No. 3, May 1971, pp. 304-314.

10. Hogg, R. V. and A. T. Graig, Introduction to Mathematical Statistics, 3rd edition, MacMillan Publishing, Inc., 1970.

11. Keringhan, B. W. and P. J. Plauger, Software Tools, Addison-Wesley Publishing Company, 1976.

12. Pearl, J., On Summarizing Data Using Probabilistic Assertions, IEEE Transactions on Information Theory, Vol. IT-23, No. 4, July 1977, pp. 459-465.

13. Pearl, J. and A. Crolette, Storage Space vs. Validity of Answers in Probabilistic Question-Answering Systems, IEEE Transactions on Information Theory, Vol. IT-26, November 1980, pp. 633-64.

14. Pierce, J. R., The Early Days of Information Theory, IEEE Transactions on Information Theory, Vol. IT-19, No. 1, January 1973, pp. 3-8.

15. Slepian, D., Information Theory in the Fifties, IEEE Transactions on Information Theory, Vol. IT-19, No. 2, March 1973, pp. 145-148

16. Tenenbaum, A. M. and M. J. Augenstein, Data Structures Using PASCAL, Prentice-Hall, Inc., 1981.

17. Viterbi, A. J., Information Theory in the Sixties, IEEE Transactions on Information Theory, Vol. IT-19, No. 3, May 1973, pp. 257-261.

18. Viterbi, A. J. and J. K. Omura, Principles of Digital Communication and Coding, McGraw-Hill, 1979.

19. Wolf, J. K., A Survey of Coding Theory: 1967-1972, IEEE Transactions of Information Theory, Vol. IT-19, No. 4, July 1973, pp. 381-388.

20. Ziv, J., Coding of Sources with Unknown Statistics--Part I: Probability of Encoding Error, IEEE Transactions on Information Theory, Vol. IT-18, No. 3, May 1972, pp. 384-389.

21. Ziv, J., Coding of Sources with Unknown Statistis--Part II: Distortion Relative to a Fidelity Criterion, IEEE Transactions on Information Theory, Vol. IT-18, No. 3, May 1972, pp. 389-394.