ED 074 769 EM 010 967

AUTHOR Knight, Joseph M., Jr.
TITLE Evaluation of a Text Compression Algorithm Against Computer-Aided Instruction (CAI) Material.
INSTITUTION Air Force Electronic Systems Div. L.G. Hanscom Field, Mass.
PUB DATE Jul 72
NOTE 37p.

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Algorithms; *Computer Assisted Instruction; *Computer Science; Mathematical Logic; *Models; Simulation
IDENTIFIERS Text Compression

ABSTRACT

This report describes the initial evaluation of a text compression algorithm against computer assisted instruction (CAI) material. A review of some concepts related to statistical text compression is followed by a detailed description of a practical text compression algorithm. A simulation of the algorithm was programed and used to obtain compression ratios for a small sample of both traditional frame-structured CAI material and a new type of information-structured CAI material. The resulting compression ratios are to 1.5: 1 for both types of material. The simulation program was modified to apply the algorithm to the lesson files of a particular frame-structured CAI subsystem used in the Air Force Phase II Base Level System. The compression in this case was found to be 1.3: 1 because some uncompressible, frame-formating bytes were present in the lesson file. The modified simulation program was also used to take letter occurrence statistics on the text being compressed. From these, a theoretical compression was calculated using a probalistic model of the compression algorithm. Theoretical compression was within two percent of measured compression, thus verifying the model's applicability. (Author)

ED 074769

ESD-TR-72-281

EVALUATION OF A TEXT COMPRESSION ALGORITHM AGAINST COMPUTER-AIDED INSTRUCTION (CAI) MATERIAL

Joseph M. Knight, Jr., Captain, USAF

July 1972

E M

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

EM 010 967

ESD-TR-72-281

EVALUATION OF A TEXT COMPRESSION ALGORITHM AGAINST COMPUTER-AIDED INSTRUCTION (CAI) MATERIAL

Joseph M. Knight, Jr., Captain, USAF

July 1972

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

FOREWORD

One of the goals of Air Force Electronic Systems Division is the development of a technology for computer-based, personnel-support systems integrated into Air Force Information Systems. These support systems are required to improve the efficiency of man-computer interactions in the host Information Systems. They are designed to provide automated on-the-job training, performance- and decision-aiding for Information Systems personnel.

Task 280104, Computer-Aided Instruction Techniques, under Project 2801, Design Methodology for Military Information Systems, was established to develop tools and techniques for computer-aided training, performance-and decision-aiding in these systems. It is also concerned with new software engineering techniques which will permit cost-effective implementation of these aids. This study relates to the latter objective.

This report, one in a series supporting Project 2801, addresses the problem of reducing the size of text files which constitute the bulk of the lesson files in the typical computer-aided instruction (CAI) systems. The approach is to simulate a practical text compression algorithm and test it against CAI lesson material. While the orientation of this study is toward CAI, the technique is generally applicable to reducing the size of text files in other systems such as data management, command and control, and intelligence data bases.

This study was performed by Captain J. M. Knight, Jr. as part of his reserve training day duties between May 1970 and September 1971, including two 2-week active duty tours. Dr. Sylvia R. Mayer, ESD/MCIT suggested the study and served as Air Force Task Scientist.

This Technical Report has been reviewed and is approved.

Sylvia R. Mayer

[illegible]

SYLVIA R. MAYER, Ph.D.
Project Scientist

MELVIN B. EMMONS, Colonel, USAF
Director, Information Systems Technology
Deputy for Command & Management Systems

ABSTRACT

This report describes the initial evaluation of a text compression algorithm against Computer-Aided Instruction (CAI) material. A review of some concepts related to statistical text compression is followed by a detailed description of a practical text compression algorithm. A simulation of the algorithm was programmed and used to obtain compression ratios for a small sample of both traditional frame-structured CAI material and a new type of information-structured CAI material. The resulting compression ratios are near 1.5 to one for both types of materials. The simulation program was modified to apply the algorithm to the lesson files of a particular frame-structured CAI subsystem used in the Air Force Phase II Base Level System. The compression in this case was found to be 1.3 to one because of the presence in the lesson file of uncompressible, frame formatting bytes. The modified simulation program was also used to take letter occurrence statistics on the text being compressed. From these, a theoretical compression was calculated using a probabilistic model of the compression algorithm. Theoretical compression was within two per cent of measured compression, thus verifying the model's applicability. The report closes with the raising of some questions and a discussion of future work.

TABLE OF CONTENTS

SECTION I

INTRODUCTION

Presently, lesson material for Computer Aided Instruction (CAI) occupies considerable disk space when the CAI system is brought on-line. For example, in the Computer-Directed Training (CODIT) subsystem of the Air Force Phase II Base Level system,(1) each 300 frame lesson is stated(2) to occupy 121,600 bytes. Even the short, "Computer Operator's Course" contains the equivalent of 14 lessons; other courses, such as the personnel course contain many more. Accordingly, the technological area of test compression is being reviewed for practical methods whereby CAI data bases may be reduced in size with only moderate computational expense.

Section II presents an elementary discussion of statistical text compression and some indication of its performance on English text. However, there also exists a simpler compression algorithm based on the practical fact that, although data characters are stored in 8-bit bytes, only about one third of the potential 256 characters are actually used in current ADP systems; the remaining two-thirds characters can be used to encode frequently occurring character pairs into single, unused characters thus obtaining data compression.

This report describes in more detail a simple, practical compression algorithm, its application to a small set of CAI data base material, and the results. Performance of the algorithm is modeled and the model is experimentally verified. In addition, a short discussion in Section VI provides guidance for future work.

SECTION II

CONCEPTS IN STATISTICAL TEXT DATA COMPRESSION

A. Bits

Data is stored in bits or in groups of bits, called bytes. One "bit" of information represents the outcome of single yes or no decision. One bit can also represent a binary state of a given situation. An ordinary room light switch can store one bit of information, e.g., "on" might mean "at home" and "off" might mean "not at home."

Groups of bits can represent more information. Two switches can represent two sequential binary decisions, i.e., four outcomes, or situation states, such as "on, on"; "on,off"; "off, on"; and "off, off". Three switches can represent eight states and, in general, N switches can represent 2^N states. A byte consisting of eight bits can represent 256 characters such as A, B, C, ... 1, 2, 3, ..., ?, $, etc. Data is generally stored one character to a byte. Nine channel magnetic data-processing tape can store 800 bytes per lineal inch of tape because the eight bits of the byte are laterally distributed across the tape, along with a ninth bit, called a parity check bit.

B. Entropy

Entropy is a property of the units, such as characters or symbols, which make up data. Entropy is a measure of the "surprisal", or information value, of a symbol. It has the units of bits/symbol and a common designation of H. A few simple examples will clarify perhaps the intuitive notion of entropy.

For instance, if it is equally likely that John is going to the seashore or the mountains this summer, and we hear that he is going to the mountains we are moderately informed, or shall we say, surprised. In this situation, the symbols "mountain" and "seashore" have for us equal information value. They are said to have equal entropy. If, on the other hand, John historically goes to the mountains nineteen summers out of twenty and we hear he is going to the mountains, we are not terribly surprised or informed. The symbol "mountain", in this case, possesses a low entropy, information value, or surprisal content. If we hear that John is going to the seashore we are quite surprised and highly informed of the happening of a low probability event. The symbol "seasore" has a high entropy, information value, or surprisal content. The entropy of a symbol is related to the priori occurrence of that symbol.

The mathematical measure of entropy H_i of the ith symbol in a data set is given by

$$H_i = -\log_2 p_i \qquad \text{(bits/symbol)} \qquad (1)$$

where P_i is the a priori probability of occurrence of the ith symbol in a data set. A symbol occurring ½ the time ($p_i = 0.5$) has an entropy of one bit/symbol. One occurring ¼ of the time ($p = 0.25$) has an entropy of two bits/symbol. One 1/8 of the time has three bits/symbol and, in general $1/2^k$ of the time has k bits/symbol entropy. Also, k may be fractional as well as integer, depending on p_i.

C. String Data

Much data is transmitted and stored in the form of strings, i.e., connected sets of alphanumeric characters, or other symbols, issuing from an information "source" and bound ultimately for an information "sink". Consider a source capable of generating four characters: A, B, C, and D, each occurring 1/4 of the time, i.e., $p_a = p_b = p_c = p_d = 0.25$. The entropy of all characters is the same and therefore the average entropy of the source is also two bits/symbol. Each character A, B, C, and D may be represented in transmission by two on-off (binary) pulses and in storage by two binarily magnetized patches on a computer tape or disk unit. But now consider a source which exhibits an unequal distribution of A, B, C, and D symbols, e.g., $p_A = 0.4$, $p_B = 0.3$, $p_C = 0.2$ and $p_D = 0.1$. Using equation (1) the entropies are calculated as $H_A = 1.32$, $H_B = 1.74$, $H_C = 2.32$ and $H_D = 3.32$, all in bits/symbol. The average (or expected value of) the source entropy H_S is given by

$$H_s = 0.4H_A + 0.3\,H_B + 0.2\,H_C + 0.1\,H_D \qquad (2)$$

The value of H_S is 1.846 bits/symbol. Note that this value is less than the 2 bits/symbol average source entropy of the "equally likely" source. A still more uneven occurrence distribution than that given above would result in a smaller source entropy.

Although it is not obvious, the above source entropy value does lead us to suspect that we can find a code, i.e., a mapping, between A, B, C, D and four groups of one or more bits each such that the average number of bits per code group is not only close to the source entropy but also is less than a straight two bits per character. This is indeed the case and the code is as follows:

A = 1 (one bit/symbol)

B = 01 (two bits/symbol)

C = 001 (three bits/symbol)

D = 000 (three bits/symbol). The coded source sequence:

1 1 0 1 1 0 0 1 0 1 1 0 0 0 · · ·

in uniquely decodeable as the original source sequence:

A A B A C B A D . . . .

Considering the probability of occurrence of A, B, C, and D we obtain an average code length of 1.9 bits/symbol.

This represents a slight compression of 1.052 relative to the original two bits/symbol. A more unequal character occurrence distribution would result in a higher compression ratio. Thus, we see that data compression can involve measuring the statistics of source symbol occurrence, designing an efficient code, and designing both an encoder and a decoder, implementable in either hardware or software.

D. Block Source Encoding

Another form of encoding is to group symbols of a source string into blocks. Consider an example where the string consists of specification of right or left-handedness and that, for our sample, right-handers outnumber left-handers by 19 to one. The probability of a right-hander, P_R, is 0.95 and the probability of a left-hander, P_L, is 0.05. Simple symbol encoding of "0" for R and "1" for L yields an average code word length of one bit/symbol. But the entropy of a binary source with .95 and .05 probabilities is only 0.286 bits/symbol. This suggests that we can do better than merely encode R and L into "0" and "1". However, it also suggests that the very best we can do is to obtain a compression of about 3.5.

Now consider coding blocks of, let us say, three symbols. We now get a new data source by grouping the old data source into blocks of three. The new data source emits eight different symbols 1, 2,. . ., 8 each representing a possible combination of three of the symbols from the old data source. The probabilities of symbol occurrence for the new data source are derivable from the probabilities of symbol occurrence from the old data source. Assuming the occurrence of any symbol is independent of the occurrence of the previous symbol we obtain, for example, the probability of symbol 3 ("RLR" - "010") from the product of probabilities

$$P_R \cdot P_L \cdot P_R = (0.95)\,(0.05)\,(0.95) = 0.04513.$$

The results are summarized in Table 1.

Table I

TABLE OF BLOCK ENCODING ENTROPIES

Symbols from Old data source	Symbols from New data source	Probability of New source symbol	H of new source symbol
000	1	.85738	.22199
001	2	.04513	4.46977
010	3	.04513	4.46977
011	4	.00237	8.72090
100	5	.04513	4.46977
101	6	.00237	8.72090
110	7	.00237	8.72090
111	8	.00012	13.02468

$H = 0.85906$ for the new source.

The entropy of the new source is 0.85906 bits/new source symbol. Notice that, on the basis of three old source symbols to one new source symbol, the entropy is also .2863 bits/old source symbol. However, now we have, with eight symbols instead of only two, more freedom to design an efficient code. There exists a technique which allows the construction of a code whose coded entropy is within one bit/symbol of the entropy of the original source. For a block length of one the code is simply one bit in length for each source symbol "0" and "1", hence, the coded source entropy is one bit/symbol which is within one bit/symbol of the source entropy, which must lie between zero and 1.00. Table 2 gives the efficient code.

TABLE 2

Table of Efficient Code for Block Length = 3

Source Symbol	Code		bability of Occurrence	"Expected value" of Symbol Length
1	1	1	.85738	.85738
2	00	2	.04513	.09026
3	010	3	.04513	.13539
4	0111	4	.04513	.18052
5	01100	5	.00237	.01185
6	011010	6	.00237	.01422
7	0110111	7	.00237	.01659
8	0110110	7	.00012	.00096
				1.10717

Average symbol length = 1.10717 bits/new source symbol.

From the above table we see that the average code word length is now 1.10717 bits/new symbol and this quantity represents three old symbols, such as RLR. This code yields a compression of 2.71 to one compared with a maximum possible compression of 3.5 to one. The use of longer blocks, and more complex codes, will result in a closer approach to the maximum possible compression figure. In this example we have assumed independence of symbol occurrence. Should there be any symbol occurrence dependence, resulting in lower entropy, block encoding will pick up this advantage also. Thus, we see that data compression not only involves measuring original source occurrence probabilities and devising efficient codes but also blocking the original source sequence into reasonable lengths, treating these as a new source, and then devising an efficient code based on the probabilities of the new source.

SECTION III

SOME TEXT COMPRESSION RESULTS

Shannon [3] gives us an estimate of the entropy of English text as a function of hc many previous letters are allowed to be known. An upper bound on compression can be calculated by dividing this entropy into the entropy of a source which puts out all letters randomly with equal probability. Table 3 gives entropies and compressions.

Table 3

Entropies and Compressions of an English Text Source Under Various Constraints

Constraint	Entropy (bits/letter)	Compression
None, 26 letters and one space equiprobable	4.76	1
Letter and space frequencies	4.03	1.18
One letter known	3.32	1.43
Two letters known	3.1	1.53
Word frequencies used	2.14	2.22

Shannon continued his investigation of english entropy beyond the point where "N-grams" of english were known. An N-gram is a histogram giving the relative frequencies of combinations of N letters. By having people predict the next letter when shown the previous L letters, Shannon was able to estimate entropies of english for constraint lengths close to 100 letters. For $10 \leq L \leq 15$ the entropy was about 1.5 bits/letter (compression = 3.17) and for $L = 100$ it was .95 bits/letter (compression = 5).

Unfortunately, compressors using constraint lengths of 100 ($\sim$20 words, or so) appear completely beyond the state-of-the-art. However, single word dictionary type compressors do appear feasible. A simulated word dictionary compression algorithm is discussed by White [4] showing results of compressions between 1.4 and 1.7 to one with a "small" dictionary and two to one with a 1000-word dictionary. For a restricted vocabulary situation, as elementary training and drill CAI may produce, we probably can take two to one as a working value for statistical word text compression. This figure compares favorably with Shannon's figure of 2.22 for word frequency compression.

Consider now the algorithm which is the object of this report. Snyderman and Hunt(5) report on a practical text compression algorithm, used at the Science Information Exchange, Smithsonian Institution, to compress the text portion of a 200,000 record on-line file from an average of 851 to 553 characters per record. This represents an implemented compression of 1.54 relative to eight bits/character, a very respectable figure.

The average of 8/1.54 = 5.2 bits/character represents a net compression of 1.245 to one relative to 6.46 bits/character for 88 equal frequency characters. This net compression lies between Shannon's theoretical compression (1.18) for an english text source when letter-space frequencies are known and the compression (1.48) when one previous letter is known. In summary, the literature indicates character text compression at around 1.5 to one and word text compression at around 2 to one.

SECTION IV

THE SNYDERMAN-HUNT COMPRESSION ALGORITHM

This section discusses more formally the Snyderman-Hunt algorithm. The algorithm was chosen to evaluate compressibility of CAI material because of its practicality, its demonstrated performance on english text and its speed. The speed of this [illegible]orithm on a 360/40 is on the order of 65-75 milli-[illegible] thousand characters, compressing or decompressing. [illegible] on the following principles.

Characters are normally stored one per 8-bit byte. With eight bits, one of $2^8 = 256$ characters can be specified by each byte. At the Scientific Information Exchange only 88 characters are used: 52 upper and lower case alphabetics, 10 numerics and 26 special characters such as comma, period, dollar sign, etc. This leaves 256-88 = 168 "unused" characters. These othe wise unused 8-bit combinations can be utilized to represent the more commonly occurring pairs of characters in the 88 used character set, thus effecting a compression.

More specifically, it is convenient to define four sets:

$$\begin{aligned} T &\triangleq \{\text{all 256 possible characters}\} \\ C &\triangleq \{\text{actual characters used}\} \\ CC &\triangleq \{\text{combining characters}\} \qquad (3) \\ MC &\triangleq \{\text{master characters}\} \end{aligned}$$

These sets are related as follow:

$$MC \subset CC \subset C \subset T \qquad (4)$$

A further set CP, for "combined pairs", can be formed of all ordered pairs of MC and CC, i.e.

$$CP = \{MC\} \quad X \quad \{CC\} . \qquad (5)$$

The members of CP can be placed in one-to-one correspondence with the difference set D defined as

$$D \triangleq \{T - C\} . \qquad (6)$$

The set of noncombining characters NC is given as

$$NC = \{C - CC\} . \qquad (7)$$

For example Snyderman and Hunt choose:

$$MC = \{\text{space}, A, E, I, O, N, T, U\} \quad (8)$$

$$CC = \{\text{space}, A \text{ through } I, L \text{ through } P, R \text{ through } W\} \quad (9)$$

The set MC has 8 members; CC has 21. The set of all combined pairs CP has 8 x 21 = 168 members which are one-to-one related to the 168 members of difference set D.

The algorithm works by examining a character in a string. If the character is a member of MC the next character is examined. If the next character is a member of CC then the two-character combined pair is coded into a single unused character and stored. If the first character is not a member of MC, it is stored as is. If the first character is a member of MC but the second one is not a member of CC then the two characters are stored individually, as is. Thus we see that compression is dependent upon both the probability of finding a master character and the conditional probability of finding a combining character given the finding of a master character. An analysis of the algorithm is presented in Appendix A.

SECTION V

EXPERIMENTS

A. Experiment One

1. Description

A computer program was written to simulate the Snyderman -Hunt algorithm. The simulation did not actually code the characters, but rather "kept score" on the number of characters that the algorithm would output for each line of input text. Compression ratio is the number of characters input divided by the number of characters output. The program, called TXTCMP, is interactive, being implemented in CPS (a subset of PL/1) for operation from a TTY or IBM 2741 terminal. TXTCMP is fed a line of text at a time and returns both line compression and total compression since the start of the program. The program listing and flow chart is reproduced in Appendix B.

The experimental material was chosen from two different types of CAI data bases: frame-structured and information-structured. The former was taken from the Computer Operator's course of reference 1, the latter from reference 6. Both are reproduced in Appendix C. The lines were entered exactly as shown in Appendix C, spaces included, from the left most character position as a reference, and the compressions were obtained. In this experiment the sets chosen by Snyderman and Hunt for master characters and for combining characters were used. The set of noncombining characters in this experiment was everything else on the IBM 2741 keyboard recognized by CPS.

In the Snyderman-Hunt application, 88 characters were valid, leaving 168 for encoding character pairs. The Snyderman-Hunt algorithm can be applied to compressing text in CPS(7) because CPS also, uses or admits in characters strings, 88 characters, leaving 168 for encoding character pairs. These results also apply to compressing text in the CODIT (Computer Directed Training) system because it is written into the Air Force Phase II Base Level System via the Burroughs B3500 COBOL language. COBOL uses $53 < 88$ characters, leaving $203 > 168$ characters for encoding character pairs. Indeed compression might be slightly better when implemented in the B3500 environment, because the 203 unused characters will accommodate 25 combining characters as opposed to only 21 in reference 5. Alternatively 9, rather than 8, master characters could be accommodated because the product of 9 master and 21 combining characters is less than the 203 characters available.

2. Results

For the frame structured material the average compression was 1.473, with individual lines (except those with a single space) ranging between 1.148 and 1.700. For the information-structured system material the average compression was 1.538 with a low of 1.261 and a high of 1.875 for individual lines. There is no particular accounting for the slight (4.4%) difference in average compression, because the spread in individual line compression is quite large in both cases with considerable overlap. From Figure 1 it is seen that the average compression settles statistically within a few lines.

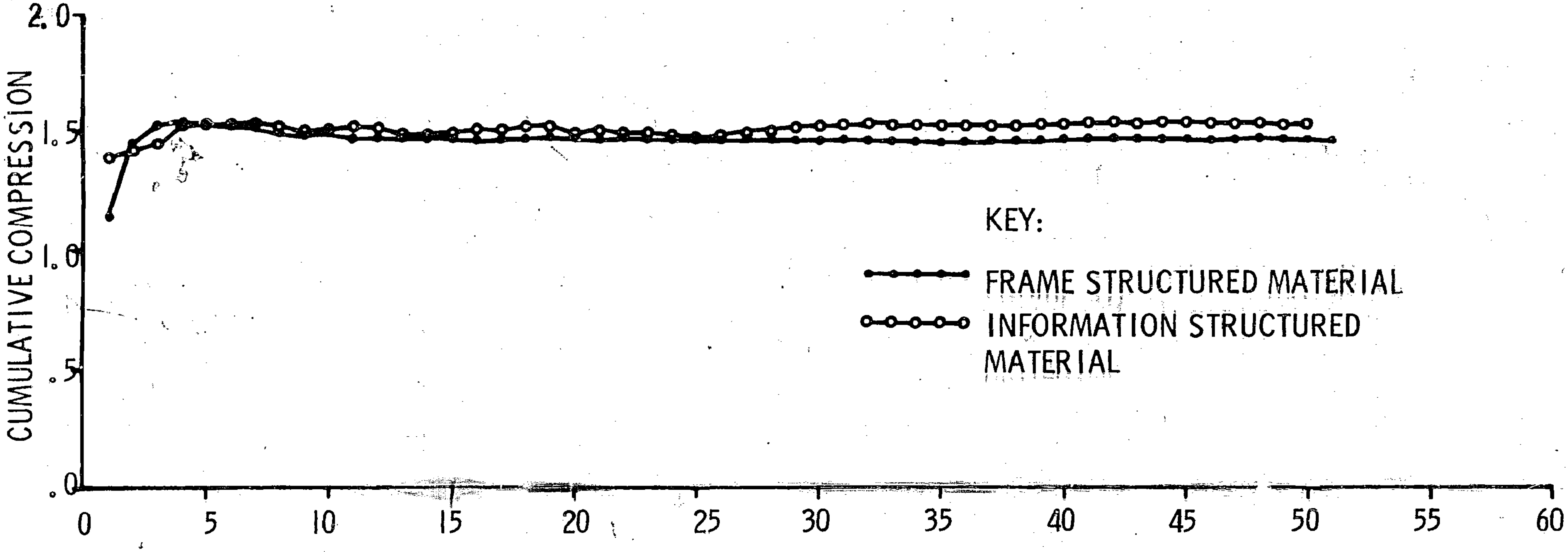


FIGURE 1 — CUMULATIVE COMPRESSION FOR COMPUTER-AIDED INSTRUCTIONAL MATERIAL

B. Experiment Two

1. Description

The objective of experiment two is to obtain an estimate of compression for the Snyderman-Hunt algorithm when applied to the actual lesson file structure of CODIT. It is found (9) that the lesson file of CODIT contains both file structure specification bytes, which are not compressible, and lesson text bytes, which are. The file structure bytes occur according to Table 4.

File Structure Bytes

Application	Number of Bytes
Frame Number	4 (per frame)
Frame Type	2 (per frame)
Frame Length	2 (per frame)
Group Number	1 (per group)
Group Length	2 (per group)
Line Number	4 (per line)
Line Length	3 (per line)

Table 4

The program TXTCMP was modified (TXCP2) to add "overhead" bytes to the compression calculation in the amount of 14 + 3 x number of groups + 7 x number of lines each time a new frame of CAI material was encountered. As an example, the CODIT printout shown in Figure 1 of Appendix C contains three frames with frame two containing three groups and six (numbered) lines.

When the CODIT CAI material was entered, only the numbered lines were entered for the compression calculation. It will be recalled that in experiment one all lines as shown in the figure were entered. The line numbers and the two spaces beyond were not entered; only the text (course author generated) to the right of this point is used. This is because all other (formatting) characters can be accounted for by the CODIT master program reading the "overhead" bytes and producing therefrom the non-text characters in the printout.

2. Results

The total CODIT subsystem compression for the material in Figures 1 and 2 of Appendix C is 1.318. While this compression is less than that obtained using all the characters in Figures 1 and 2, it is a more realistic value because the CODIT file structure "overhead" bits are taken into account. Also, it is a conservative (low) value because the frames in the experimental set have very little expository text material. The frames are largely for questioning the trainee rather than for instructing him. One can reasonably expect an experimental set containing a mix of questioning frames and instructing frames to yield a higher compression, Even so, the 1.318 figure has useful implications. In the CODIT subsystem it means reducing each 121,600 byte lesson file by about 28,000 bytes or, alternatively, putting 30% more lessons on disk for the same CAI file allocation in the Air Force Phase II Base Level System. Putting more lessons on-line gives increased daily flexibility to the OJT/CAI program. Using less disk for CAI increases the chances for its acceptance since it leaves adequate disk space for the other functional areas, such as personnel, finance and civil engineering.

C. Experiment Three

1. Description

The objective of experiment three is to verify the analytical model of the Snyderman-Hunt algorithm developed in Appendix A. The essence of the model is equation (7), Appendix A, which predicts compression on the basis of p_1, the probability of a master character occurring, and p_1p_2 the joint probability of both a master and a combining character occurring together. Should the model be verified to an engineering degree of accuracy, it would then be possible to select more easily optimum master and combining characters sets because p_1 is simply related to single letter and space relative occurrences in english and p_1p_2 is also simply related to double letter and space relative occurences. When TXCMP was developed into TXTCP2, provision was made to measure p_1 and p_2 and p_1p_2 on the text portion of the experimental material. A theoretical, or predicted, text compression was calculated. The experimental material used was the text portion (1003 characters) of Figures 1 and 2 of Appendix C.

2. Results

Using the text material only, i.e., no CODIT subsystem "overhead" bytes considered, it is found that the material of Figures 1 and 2, Appendix C, yield $P = .566$, $p_1p_2 = .531$ and a theoretical compression of 1.513. This value compares quite well (within 2%) of the experimentally measured text compression, 1.530. Furthermore, examination of cumulative measured text compression and cumulative theoretical text compression as it builds up on a line-by-line basis shows that the compression predicted by equation (7) of Appendix A is stable and always within 2.5 per cent, thus indicating a valid model for the Snyderman-Hunt algorithm.

SECTION VI

CONCLUSIONS, QUESTIONS AND RECOMMENDATIONS

Based on these results, three major conclusions follow:

1. A working figure of 1.5 may be taken for the practical compression of CAI text material.

2. When frame formatting overhead bytes are taken into account in a typical CAI system, the compression figure becomes, conservatively, 1.3 to one.

3. It is possible to adequately model the Snyderman-Hunt algorithm and predict compression performance within a few per cent, based on text statistics.

Given these conclusions several timely questions may be raised:

How can the Snyderman-Hunt algorithm be optimally applied to CODIT which is now being implemented Air Force-wide? Where would the compression and decompression algorithm be inserted into the CODIT system flow diagram (pg. 50 of reference 1)? Can you patch a B3500 assembly language compression decompression algorithm into a compiled COBOL CODIT program? Given that COBOL uses only 53 characters, what is now the optimum master and combining character sets? What is the dollar saving in reduced disk files and magnetic tapes? By how much is this dollar saving offset by the 75-odd microsecond per character CPU time cost? The dollar saving questions can be approached in two ways:

1. By taking gross costs from the current B3500 Base Level System installation with estimates of CAI file space, CAI character throughput, and B3500 speed for compressing and decompressing, it is possible, in-house, to arrive at a rough estimate of dollar saving.

2. By putting this problem to industry as a contracted study wherein the contractor designs an optimal compression system based on extensive CAI data base material, does a preliminary system design around current or projected hardware, and calculates relative costs of going compressed and uncompressed within the system.

It is recommended that (1) above, be accomplished and, based on the outcome, (2) be considered, perhaps as part of contract definition for Air Force systems beyond B3500. It is also recommended that text compression be considered if CODIT is rewritten in JOVIAL for DAFCCS application. Finally it is recommended that the Snyderman-Hunt algorithm be experimentally applied to other Air Force textual data bases, such as intelligence.

APPENDIX A

Analysis of the Snyderman-Hunt Text Compression Algorithm

Consider a string of N characters. As a character is examined to see if it is a master character, there is the possibility that either one or two characters will be read in. Let p_1 be the probability that the character examined is a master character and $1-p_1$ the probability it is not. If the character is a master character, then a second character will be read in; if it is not, then only the single character is read in, and the cycle repeated. The expected number of characters input, per cycle, is given by

$$\begin{aligned} ECI &= 2\ (p_1) + 1\ (1-p_1) \\ &= 1 + p_1\ . \end{aligned} \tag{1}$$

For a string of N characters the number of read cycles R is given by

$$R = \frac{N}{1 + p_1}\ . \tag{2}$$

When a master character is found, with probability p_1, two possibilities exist: the next character will be a combining character, or it will not. Let p_2 be the probability that the next character will be a combining character and $1 - p_2$ that it will not. If the second character is a combining character, it will be combined with the master character and only one character will be read out. If the second character is not a combining character, then two characters will be read out. If the first character is not a master character only one character will be read out. These rules lead to the expected number of characters output per cycle, being given by

$$\begin{aligned} ECO &= 1\ (p_1\ p_2) + 2\ (p_1\ (1-p_2)\ )\ +\ 1\ (1-p_1) \\ &= 1 + p_1 - p_1\ p_2 \end{aligned} \tag{3}$$

The expected number of characters read out NO, per line of N characters in, is given by

$$NO\ =\ R\ \ (ECO) \tag{4}$$

$$NO\ =\ R\ \ (1 + p_1 - p_1\ p_2)\ . \tag{5}$$

Compression C is defined as the number of characters N in the line divided by the number of characters NO read out from the line processing, i.e.

$$C = \frac{N}{NO} . \qquad (6)$$

Substituting previous work in the above, we relate expected compression to the probabilities p_1 and p_2.

$$\begin{aligned} C &= R\ (1 + p_1) \ / \ R\ (1 + p_1 - p_1\ p_2) \\ &= \frac{1 + p_1}{1 + p_1 - p_1\ p_2} \end{aligned} \qquad (7)$$

Note that if all first read characters are master characters, $p_1 = 1$, and if all second read characters are combining characters, $p_2 = 1$, then C is a maximum and equal to 2. On the other hand, if no master characters occur, $p_1 = 0$, then compression is at a minimum and equal to unity. Since p_1 is the probability of finding a master character p(MC) and p_2 is the probability p(CC/MC) of finding a combining character, given a master character, we see that $p_1 p_2$ is the joint probability p(MC,CC) of finding a master character and a combining character together. Both p(MC) and p(MC,CC) can be experimentally determined for a given data base, such as english, once a table of first and second order occurrences is compiled and the sets of master characters and combining characters are defined. The sets can be adjusted, within the constraints given in the text, to maximize the expected compression.

APPENDIX B

TXTCMP Program Listing

Note 1: The program operates by working its way (via POINT) through the LINE of text, character by character. If a master character is not found, both the compressed and uncompressed bit count are augmented by one byte: if a master character is found, the next character is tested for being a combining character. If the next character is a combining character, the compressed bit count is augmented by one byte and the uncompressed bit count by two bytes, otherwise both compressed and uncompressed bit counts are augmented by two bytes. An isolated master character at the end of LINE will be so identified (program line 350) and cause augmentation of both compressed and uncompressed bit counts by one byte. Success of the end of line test initiates printout.

Note 2: Program line 426 is not essential to operation; it merely prints the value of POINT occasionally to let you know the program is functioning during the wait between line input and compression printout.

Note 3: Variable listing

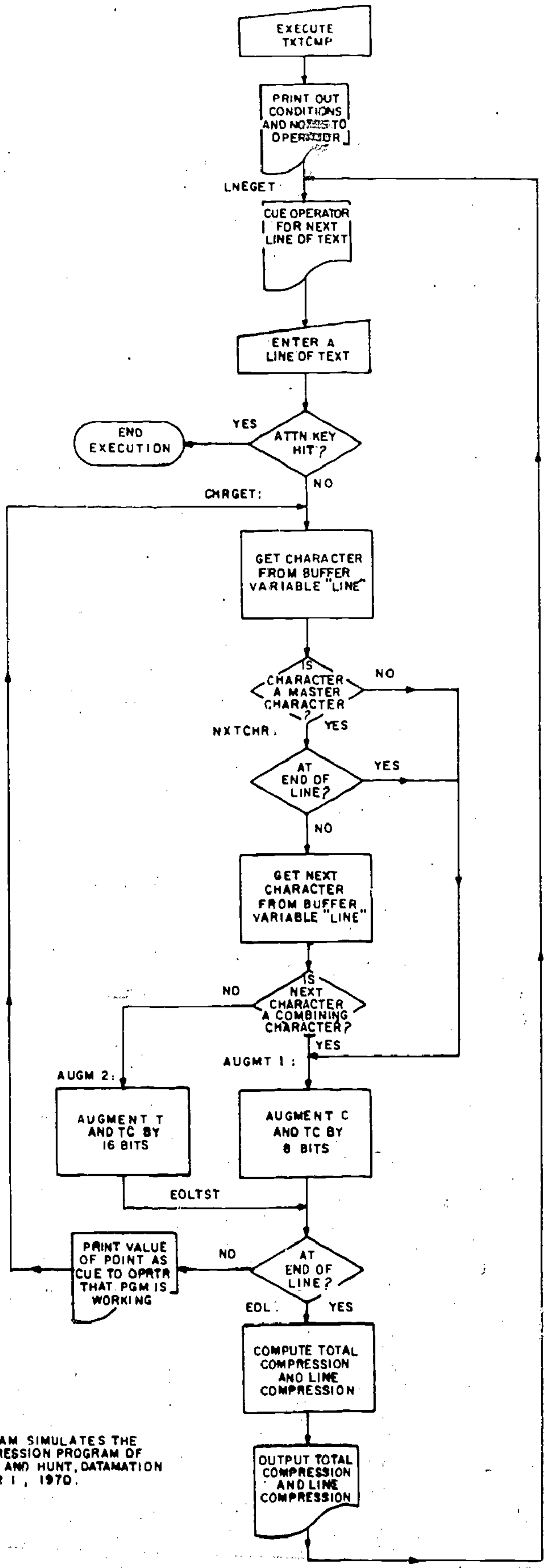
Variable	Explanation
M(I).	.Master Character array
CC(I)	.Combining character array
TUC	.Number of bits, uncompressed, from beginning of program
TC	.Number of bits, compressed, from beginning of program
LINE.	.Character variable containing a line of text
UC	.Number of bits, uncompressed, in a given line
C	.Number of bits, compressed, in a given line
POINT	.A text pointer variable
TEST1	.A character variable containing one character being tested to see if it is a master character
I	.A general indexing variable

TEST2	A character variable containing one character beomg tested to see if it is a combining character.
TOTCMP.	Total Compression since beginning of program
LNECMP.	Compression of the given line above

Note 4: Label Listing

Label	Explanation
TXTCMP.	The name of the program: "Text Compression"
LNEGET.	Get a new line of text
CHRGET.	Get a new character from the line
NXTCHR.	Get the next character (following an identified master character)
AUGMT2	Augment the bit count by 2 bytes (16 bits)
AUGMT1.	Augment the bit count by 1 byte (8 bits)
EOLTST	End of line test
EOL	End of line

[IC-35,873]



EXECUTE TXTCMP
PRINT OUT CONDITIONS AND NOTES TO OPERATOR
LNEGET
CUE OPERATOR FOR NEXT LINE OF TEXT
ENTER A LINE OF TEXT
ATTN KEY HIT?
YES
END EXECUTION
NO
CHRGET:
GET CHARACTER FROM BUFFER VARIABLE "LINE"
IS CHARACTER A MASTER CHARACTER?
NO
YES
NXTCHR:
AT END OF LINE?
YES
NO
GET NEXT CHARACTER FROM BUFFER VARIABLE "LINE"
IS NEXT CHARACTER A COMBINING CHARACTER?
NO
YES
AUGM 2:
AUGMT 1:
AUGMENT T AND TC BY 16 BITS
AUGMENT C AND TC BY 8 BITS
EOLTST
PRINT VALUE OF POINT AS CUE TO OPRTR THAT PGM IS WORKING
NO
AT END OF LINE?
EOL:
YES
COMPUTE TOTAL COMPRESSION AND LINE COMPRESSION
OUTPUT TOTAL COMPRESSION AND LINE COMPRESSION

THIS PROGRAM SIMULATES THE TEXT COMPRESSION PROGRAM OF SNYDERMAN AND HUNT, DATAMATION DECEMBER 1, 1970.

FLOW CHART FOR TXTCMP

FIGURE B-1 FLOW CHART FOR TXTCMP

```
TXTCMP: PROCEDURE ;
        /*THIS PROGRAM SIMULATES THE TEXT COMPACTION ALGORITHM OF*/;
        /*SNYDERMAN AND HUNT, DATAMATION, DEC 1, 1970.*/;
        PUT LIST(' ');
        PUT LIST('EXECUTING TEXT COMPRESSION.');
        PUT LIST('PLEASE NOTE: ENTER ALL CHARACTERS IN UPPERCASE.');
        PUT LIST('ALSO NOTE: LIMIT LINE TO 70 CHARACTEPS.');
        PUT LIST('HIT ATTN ON IBM 2741 TERMINAL OR BREAK ON TTY TO END PROGRAM.');
        PUT LIST(' ');
        DECLARE M(8) CHAR(2), LINE CHAR(70) VAR;
        DECLARE CC(21) CHAR(1);
        DECLARE  TEST1 CHAR(1), TEST2 CHAR(1);
        M(1)=' ';
        M(2)='A';
        M(3)='E';
        M(4)='I';
        M(5)='O';
        M(6)='N';
        M(7)='T';
        M(8)='U';
        CC(1)=' ';
        CC(2)='A';
        CC(3)='B';
        CC(4)='C';
        CC(5)='D';
        CC(6)='E';
        CC(7)='F';
        CC(8)='G';
        CC(9)='H';
        CC(10)='I';
        CC(11)='L';
        CC(12)='M';
        CC(13)='N';
        CC(14)='O';
        CC(15)='P';
        CC(16)='R';
        CC(17)='S';
        CC(18)='T';
        CC(19)='U';
        CC(20)='V';
        CC(21)='W';
        TUC=0;
        TC=0;
LNEGET: PUT LIST('LINE');
        READ INTO(LINE) ;
        UC=0;
        C=0;
        POINT=1;
CHRGET: TEST1=SUBSTR(LINE,POINT,1);
        TUC=TUC+8;
        UC=UC+8;
        DO I=1 TO 8;
        IF TEST1=M(I) THEN GO TO NXTCHR;
        END ;
        GO TO AUGMT1;
NXTCHR: IF POINT=LENGTH(LINE) THEN GO TO AUGMT1;
        POINT=POINT+1;
        TEST2=SUBSTR(LINE,POINT,1);
        TUC=TUC+8;
        UC=UC+8;
        DO I=1 TO 21;
        IF TEST2=CC(I) THEN GO TO AUGMT1;
        END ;
AUGMT2: C=C+16;
        TC=TC+16;
        GO TO EOLTST;
AUGMT1: C=C+8;
        TC=TC+8;
EOLTST: IF POINT=LENGTH(LINE) THEN GO TO EOL;
        POINT=POINT+1;
        IF POINT/6=TRUNC(POINT/6) THEN PUT LIST(POINT);
        GO TO CHRGET;
EOL:    TOTCMP=TUC/TC;
        LNECMP=UC/C;
        PUT LIST(' ');
        PUT LIST('LINE COMPRESSION');
        PUT LIST(LNECMP);
        PUT LIST('TOTAL COMPRESSION');
        PUT LIST(TOTCMP);
        PUT LIST(' ');
        GO TO LNEGET;
        END TXTCMP;
```

FIGURE B-2 CPS LISTING OF PROGRAM
TXTCMP: "TEST COMPRESSION"

APPENDIX C

EXPERIMENTAL MATERIAL

Reproduction of frame-structured and information-structured CAI material. All parts of all lines containing one or more characters constitutes the experimental set for experiment one. Only the text portions of numbered lines in Figures C-1 and C-2 constitute the experimental set for experiments 2 and 3.

```
LESSON 0007000 DATE WRITTEN 160569 PAGE 1

        FRAME     1.0     TYPE     M1     LABEL    000700

        G.2 TEXT

          1.0                       VIT PROGRAMMING LANGUAGES??
          2.0    DO YOU WANT TO TRY THE LESSON ON PROGRAMMING LANGUAGES
          3.0    OR DO YOU THINK YOU CAN SKIP IT?

        G.3 ANSWERS

          1.0    A+I WILL TRY THE LESSON ON PROGRAMMING LANGUAGES.
          2.0    B+I THINK I KNOW ENOUGH TO SKIP IT.

        G.4 ACTIONS

          1.0    A F:FINE.  LET'S BEGIN.  8:31
          2.0    B F:WE'LL GIVE YOU A LITTLE TEST JUST TO MAKE SURE.

        FRAME    2.0      TYPE     Q1     LABEL

        G.2 TEXT

          1.0    WHAT DOES COBOL STAND FOR?

        G.3 ANSWERS

          1.0    0 SET KEYWORD ON
          2.0    0 SET PHONETIC ON
          3.0    0 SET ORDER ON
          4.0    A+COMMON BUSINESS ORIENTED LANGUAGE

        FRAME    3.0      TYPE     Q1     LABEL

        G.2 TEXT

          1.0    WHAT DOES FORTRAN STAND FOR?

        G.3 ANSWERS

          1.0    A+FORMULA TRANSLATION
```

FIGURE C-1 CODIT SUBSYSTEM FRAME
STRUCTURED CAI MATERIAL

```
LESSON 000700 DATE WRITTEN 160569 PAGE  2

      FRAME    4.0      TYPE     Q1     LABEL

      G.2 TEXT

        1.0    WHAT DOES RPG STAND FOR?

      G.3 ANSWERS

        1.0    A+REPORT PROGRAM GENERATOR

      FRAME    5.0      TYPE     M1     LABEL

      G.2 TEXT

        1.0    WHAT IS MADE UP OF 1'S AND Q'S?

      G.3 ANSWERS

        1.0    A+MACHINE LANGUAGE
        2.0    B PROCEDURE-ORIENTED LANGUAGE
        3.0    C RPG LANGUAGE
        4.0    D OCTAL LANGUAGE
        5.0    E NONE OF THE ABOVE

      FRAME    6.0      TYPE     Q1     LABEL

      G.2 TEXT

        1.0    WHAT DO YOU CALL MACHINE-SPECIFIC INSTRUCTIONS USED BY A
        2.0    PROGRAMMER SPECIALIST TO REPRESENT EACH MACHINE OPERATION?
        3.0      (THE WORD 'MACHINE' SHOULD NOT BE INCLUDED)

      G.3 ANSWERS

        1.0    A+MNEMONIC
        2.0    B+SYMBOLIC
        3.0    C+SYMBOLIC CODE

      FRAME    7.0      TYPE     D1     LABEL

      G.2 CONDITIONS

        1.0    IF GQ 2 WRONG 2-6 F: YOU'RE OFF TO A BAD START.  YOU'D BETTER
        2.0    F:TRY THE LESSON. B:MOD7
```

FIGURE C-2 CODIT SUBSYSTEM FRAME
STRUCTURED CAI MATERIAL

```
(RPAQQ LATITUDE (((ON LATITUDE)
      (DET THE DEF 2))
    NIL
    (SUPERC NIL (DISTANCE NIL ANGULAR (FROM NIL
            EQUATOR)))
    (SUPERP (I 2)
      LOCATION)
    (VALUE  (I 2)
      (RANGE NIL -90 90))
    (UNIT (I 2)
      DEGREES)))


(RPAQQ ARGENTINA (((XN ARGENTINA)
      (DET NIL DEF 2))
    NIL
    (SUPERC NIL COUNTRY)
    (SUPERP (I 6)
      SOUTH/AMERICA)
    AREA (I 2)
      (APPROX NIL/120000000
    (LOCATION NIL SOUTH/AMERICA (LATITUDE (I 2)
         (RANGE NIL -22 -55))
      (LONGITUDE (I 4)
         (RANGE NIL -57 -71))
      (BORDERING/COUNTRIES (I 1)
         (NORTHERN (I 1)
            BOLIVIA PARAGUAY)
         (EASTERN (I 1)
            (($L BRAZIL URUGUAY
              NIL
              (BOUNDARY NIL URUGUAY/RIVER)))

     (CAPITAL (I 1)
       BUENOS/AIRES)
     (CITIES (I 3)
       (PRINCIPAL NIL ($L BUENOS/AIRES CORDOBA ROSARIO
              MENDOZA LA/PLATA TUCUMAN)))
     (TOPOGRAPHY (I 1)
       VARIED
       (MOUNTAIN/CHAINS NIL (PRINCIPAL NIL ANDES
              (LOCATION NIL (BOUNDARY NIL (WITH NIL
                     CHILE)))
            (ALTITUDE NIL (HIGHEST NIL ACONCAGUA
                   (APPROX NIL 22000))))
          (SIERRAS NIL (LOCATION NIL ($L CORDOBA
                   BUENOS/AIRES))))
       (PLAINS NIL (FERTILE NIL USUALLY)
          (($L EASTERN CENTRAL)
            NIL PAMPA)
          (NORTHERN NIL CHACO)))
```

FIGURE C-3 THE UNITS FOR LATITUDE AND ARGENTINA (FRAGMENTS) IN SCHOLAR, AN INFORMATION STRUCTURED CAI SYSTEM

REFERENCES

1. No author, "The Development of a Computer-Directed Training Subsystem and Computer Operator Training Material for the Air Force Phase II Base Level System", ESD-TR-70-27, 30 November 1969.

2. Butler, A. K., et al, "Training and Design Requirements for an Air Force Computer-Aided Training Subsystem for the World-Wide Military Command and Control System", ESD-TR-68-415, 30 September 1968.

3. Shannon, C. E., "Prediction and Entropy of Printed English", Bell System Technical Journal, Vol. 30, pp. 50-64, January 1951.

4. White, W. W., "Printed English Compression by Dictionary Encoding", Proc. IEEE, Vol. 55, No. 3, March 1967, pp. 390-396.

5. Snyderman, M. and Hunt, B., "The Myriad Virtues of Text Compaction", Datamation, Vol. 16, No. 16, December 1, 1970, pp. 36-40.

6. Carbonnell, J. R. and Collins, A.M., "Mixed Initiative Systems for Training and Decision Aid Applications", ESD-TR-70-373, November 1970, pg. 51, units for LATITUDE and ARGENTINA.

7. No author, "Conversational Programming System, Terminal User's Manual", 1 August 1969, The MITRE Corporation, Bedford, Mass. 01730.

8. No author, "Burroughs B2500/B3500 Information Processing System COBOL Reference Manual", May 1969, Burroughs Corporation, Detroit, Mich. 48232, page 1-1.

9. Butler, A. K., et al, "Performance/Design Requirements and Detailed Technical Description for a Computer-Directed Training Subsystem for Integration into the Air Force Phase II Base Level System", ESD-TR-68-30, June 1968, pages 16, 26.

Security Classification

DOCUMENT CONTROL DATA - R & D

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

1. ORIGINATING ACTIVITY (Corporate author) Deputy for Command and Management Systems Hq Electronic Systems Division (AFSC) L G Hanscom Field, Bedford, Mass. 01730	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP N/A

3. REPORT TITLE
EVALUATION OF A TEXT COMPRESSION ALGORITHM AGAINST COMPUTER-AIDED INSTRUCTION (CAI) MATERIAL

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
None

5. AUTHOR(S) (First name, middle initial, last name)
Joseph M. Knight, Jr., Captain, USAF

6. REPORT DATE July 1972	7a. TOTAL NO. OF PAGES 29	7b. NO. OF REFS 9
8a. CONTRACT OR GRANT NO. IN-HOUSE b. PROJECT NO. 2801 c. TASK NO. 280104 d.	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-72-281	
	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Deputy for Command and Management Systems Hq Electronic Systems Division (AFSC) L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

This report describes the initial evaluation of a text compression algorithm against Computer-Aided Instruction (CAI) material. A review of some concepts related to statistical text compression is followed by a detailed description of a practical text compression algorithm. A simulation of the algorithm was programmed and used to obtain compression ratios for a small sample of both traditional frame-structured CAI material and a new type of information structured CAI material. The resulting compression ratios are near 1.5 to one for both types of materials. The simulation program was modified to apply the algorithm to the lesson files of a particular frame-structured CAI subsystem used in the Air Force Phase II Base Level System. The compression in this case was found to be 1.3 to one because of the presence in the lesson file of uncompressible, frame formatting bytes. The modified simulation program was also used to take letter occurrence statistics on the text being compressed. From these, a theoretical compression was calculated using a probabilistic model of the compression algorithm. Theoretical compression was within two per cent of measured compression, thus verifying the model's applicability. The report closes with the raising of some questions and a discussion of future work.

DD FORM 1473 1 NOV 65

Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
data compression						
text compression						
CAI						
computer aided instruction						
information theory						
probability						
mathematical model						

Security Classification