ED 151 016 IR 005 739

AUTHOR	Stone, Maureen
TITLE	Application of Data Compression Techniques to the PLATO IV Communication System.
INSTITUTION	Illinois Univ., Urbana. Computer-Based Education Lab.
SPONS AGENCY	Advanced Research Projects Agency (DOD), Washington, D.C.
PUB DATE	Jul 77
CONTRACT	DAHC-15-73-C-0077
NOTE	57p.
EDRS PRICE	MF-$0.83 HC-$3.50 Plus Postage.
DESCRIPTORS	*Computer Assisted Instruction; Computer Graphics; Computer Oriented Programs; Cost Effectiveness; Data Processing; *Display Systems; *Electronic Data Processing
IDENTIFIERS	*Data Compression; PLATO IV

ABSTRACT

This paper presents a study on the effects of various data compression methods from the viewpoint of central computer to terminal communications on a large graphics oriented timesharing system, PLATO IV. The desired goal is to increase terminal display speed without significant increase in the transmission error rates. While the major emphasis in this paper is on text transmission, some discussion of other display functions is included. Chapters provide (1) a description of the PLATO IV architecture and communications system; (2) a review of two projects involving processor based terminals; (3) an explanation of how text is currently transmitted, followed by an analysis of the average number of bits/character obtained by this method; (4) an introduction to the theoretical background for variable length or Huffman coding; (5) projected savings and overhead involved in the use of word lists to reduce the average number of bits/character used to represent text; and (6) conclusions, suggestions for future research, and discussions of projects involving text compression and other methods of improving display speed. (Author/DAG)

Application of Data Compression Techniques to the PLATO® IV Communication System

Maureen Stone

Computer-Based Education Research Laboratory,
University of Illinois, Urbana, Illinois

This manuscript was prepared with partial support from the Advanced Research Projects Agency (U.S. Army Contract DAHC-15-73-C-0077) and the University of Illinois at Urbana-Champaign.

ACKNOWLEDGMENT

I would like to express my appreciation to all those who have helped me with this project; particularly to Professor Roger Johnson for providing both direction and encouragement.

For providing both technical advice and a forum for ideas, I would like to acknowledge members of the Hardware Research Group; Doug Brown, Kevin Gorey, Paul Lamprinos, Todd Little, Jim Opperheimer, and Scott Weikart. Similarly, I would like to acknowledge members of the PLATO IV systems staff; Dave Andersen, Rick Blomme, Bob Rader, Bruce Sherwood, and Paul Tenczar.

For helping me produce this paper, I would like to thank members of the CERL staff, and Chris Fleener.

TABLE OF CONTENTS

1. INTRODUCTION

With the development of various mini and micro-processor based terminals, it is appropriate to re-examine the question of communication requirements between the central computer and the terminal in the PLATO system. With a processor in the terminal, it is reasonable to reconsider decoding algorithms that were previously too expensive in terms of terminal hardware.

This paper presents a study on the effects of various data compression methods from the viewpoint of central computer to terminal communications on a large graphics oriented timesharing system, PLATO IV. The desired goal is to increase terminal display speed without significant increase in the transmission error rates. While the major emphasis in this paper is on text transmission, some discussion of other display functions is included. The paper is organized into seven chapters.

Chapter 2 provides a general description of the PLATO IV architecture and communications system.

Chapter 3 is a review of two projects involving processor based terminals, one using a 16 bit mini-computer, and one using an 8 bit micro-processor.

Chapter 4 gives a detailed explanation of how text is currently transmitted, followed by an analysis of the average number of bits/character obtained by this method. Three areas for improvement are described.

The theoretical background for variable length or Huffman coding is introduced in Chapter 5. Both the projected gains, and some design considerations for implementation on PLATO IV are given.

The use of word lists is a method successfully used in other applications to obtain a significant reduction in the average number of bits/character used to represent text. Both the projected savings using this method and the overhead involved are described in Chapter 6.

Chapter 7 contains both conclusions and suggestions for future research. Projects involving text compression and other methods of improving display speed are discussed.

2. PLATO IV SYSTEM ARCHITECTURE

2.1 Central Computer Architecture

The PLATO IV computer-based education system consists of a large central computer, the Control Data Corporation Cyber 73-24, with more than 900 graphics terminals connected to it. (5,10) The Cyber 73-24 is a dual processor system with the two central processing units (CPU's) connected to the same central memory (Figure 2.1). Two million 60 bit words of extended core storage (ECS) are directly coupled to central memory by high speed block transfers. The ten peripheral processing units (PPU's), which are small, programmable processors, can access both ECS and central memory. Most input/output information is transferred through the PPU's to buffers in ECS. In this way, ECS becomes the central transfer point for all data. (1)

2.2 Communications Architecture

The communication system for the terminals is unusual, as can be seen in Figure 2.2. The data rate is asymmetrical, with the output rate to the terminal being 32 times faster than the input rate. Standard television equipment and voice grade phone lines were used to give the lowest possible cost.

Information for a given terminal is sent from the central computer through a PPU to the Network Interface Unit (NIU). There it is interleaved with the information for all other terminals and transmitted as a video signal. At a particular location, the site controller selects the data for the terminals at the site. The information is

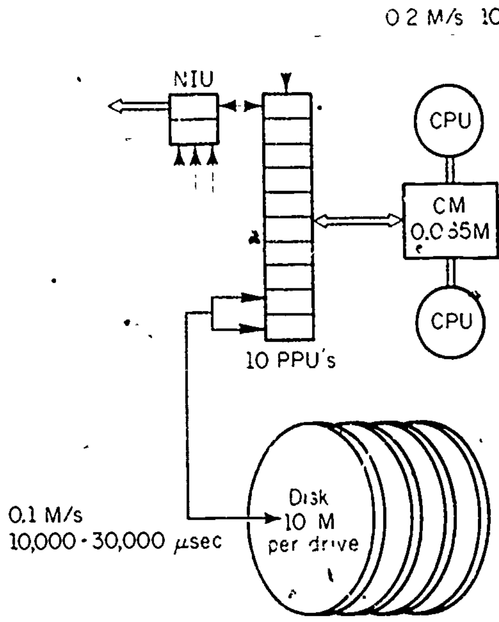


Figure 2.1. PLATO IV computer architecture, showing memory sizes in 60-bit words, transfer rates in 60-bit words/sec, and access times in microseconds. M = million, CPU = central processing unit, PPU = peripheral processing unit, NIU = network interface unit, CM = central memory, ECS = Extended Core Storage. Programs and data are swapped between CM and ECS. Conventional disk drives provide permanent storage for programs and data. The basic computer is a Control Data Corporation Cyber 73-24.

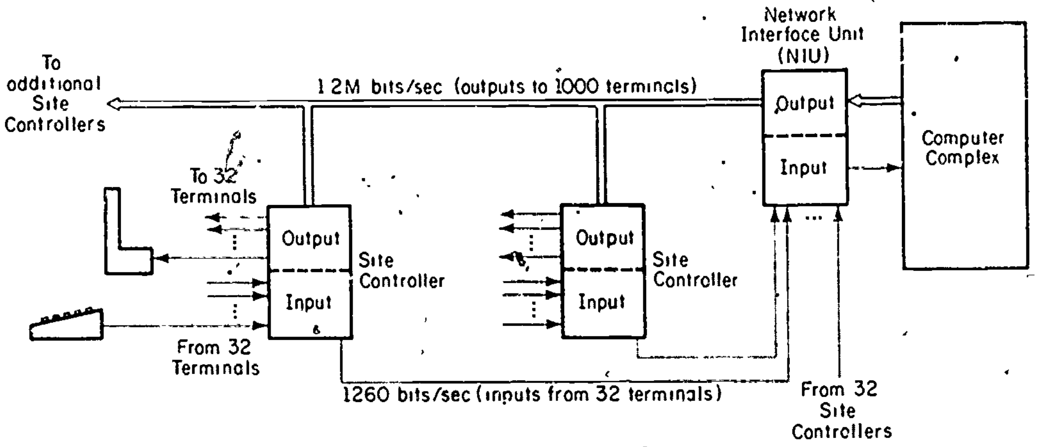

Network Interface Unit (NIU)
To additional Site Controllers
1.2M bits/sec (outputs to 1000 terminals)
Output
Input
Computer Complex
To 32 Terminals
Output
Input
Site Controller
Output
Input
Site Controller
From 32 Terminals
1260 bits/sec (inputs from 32 terminals)
From 32 Site Controllers

Figure 2.2. Communications hardware configuration.

separated and sent to the appropriate terminal over a voice grade phone line. The limiting channel is the phone line; therefore the data rate to the terminal is usually given as the rate along this phone line, or 1260 baud.

Input, which is usually in the form of key presses, is transferred to the site controller along the reverse channel of the phone line. The input data for all terminals at the site is transmitted over a single 1260 baud line back to the central computing system. Since there can be up to 32 terminals at one site, the data rate back is up to 32 times slower than the data rate out. More detailed information on the communications system is given in (1).

All terminals on the PLATO IV system use the same information protocol for output which is unique to the PLATO system. Every 16.7 ms, a 21 bit parcel containing 19 bits of information, 1 bit parity, and 1 start bit, is received by every terminal. This is either information from the central computer or an all zero NOP generated by the network interface unit. This length format was chosen to accommodate the 9 bits x and 9 bits y needed for panel addressing. An extra bit was needed to distinguish data from control words. Because the system is synchronous, every 16.7 ms a frame must be generated by the central site, consisting of one 20 bit parcel of information for each terminal which has output pending. The output is originally generated by a running program or "lesson" (Figure 2.3). The bulk of the lesson is resident in ECS, with only a small logical block or "unit" resident in central memory. Output is encoded by the Executor

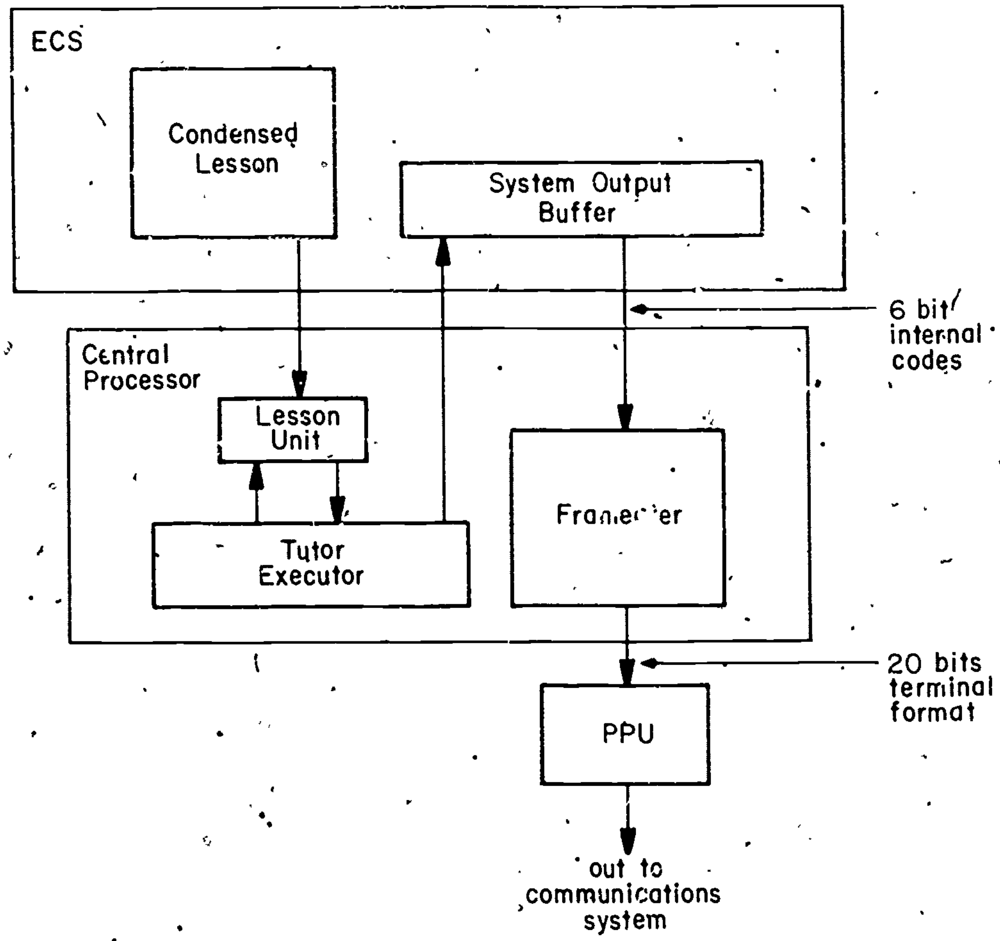


Figure 2.3. PLATO IV output software configuration.

and placed in the system output buffer. However, the information in this buffer is in a generalized, terminal independent form, and not in the 20 bit format required by the PLATO IV terminal. The conversion to terminal format is handled by a separate program, which also periodically creates the frame described above and sends it, through a PFU, to the communications system. This same program, called the Frameater, also keeps track of each terminal's current state to avoid sending redundant information. While 20 bits/parcel are sent by the Frameater to the NIU, parity is actually generated by the communications hardware.

3. REVIEW OF WORK WITH PROGRAMMABLE TERMINALS

The current PLATO IV terminal consists of a 512 x 512 matrix plasma display, keyset, and a touch input device called a touch panel. Available display functions are line drawing, character plotting, and single point plotting. There are 252 available characters, ½ of them dynamically user-programmable from the central computer. Most of the current terminals realize these functions through a MSI/TTL design currently manufactured by Magnavox. (2).

However, it has been recognized throughout the history of PLATO IV that it would be valuable to use a processor in the terminal. During the procurement of the first PLATO IV terminals, a processor-based design was considered, but rejected on the basis of cost (11). More rec tly, with the evolution of low-cost LSI micro-processor technology, consideration has again been given to processor-based PLATO terminals. This concept has been explored through two projects at CERL.

In 1972, a project directed by R. L. Johnson was started using a Digital Equipment Corporation PDP 11/05 as the basis for a programmable or "intelligent" terminal. Besides the use of the processor, this terminal differed from the standard one because it used a version of the plasma panel which could operate on 16 display points in parallel. This modified panel was therefore capable of a display speed up to 16 times faster than the standard panel. Results of this project are published elsewhere (3).

The most interesting feature of this programmable terminal was the ability to combine high speed display with the flexible presentation.

structure of the PLATO IV system. That is, the PLATO lesson could determine the basic design of the display, and the mini-computer could help to get it up on the screen quickly. For example, a major difficulty with display devices such as the plasma panel which have inherent memory is that to erase an area takes as long as it does to write it, with the exception of the full screen erase. For the standard system, due to the synchronized communication and the speed of the plasma panel, area erasure is limited to the maximum character plotting rate of 180, 8 x 16 characters per second. For the programmable system, a terminal function called "block erase" was defined that, given opposite corners of a rectangle, would erase the area. Using the parallel panel, this achieves impressive speeds. Other defined functions for the system include circle generation, rectangular and circular shaded areas, and large sized characters. For more specialized displays, a protocol was defined for loading and calling PDP-11 subroutines from PLATO lessons. Within the PDP-11, system subroutines were available for most display functions. However, it is impossible to match the ease of designing a display as is done on PLATO with subroutines for a mini-computer assembly language. Both the language and the utilities are lacking. But it is possible to locally store the 20 bit parcels provided by the PLATO generated display, feed them back through the terminal simulator, and see a large increase in display speed. This process, called image trapping, has been successfully used to plot most of the displays in a group of highly interactive medical-information system lessons. The major draw-back is the large amount of storage needed. For more than a few full page displays, it is necessary to use an auxiliary storage medium such as a floppy disk. This project is continuing; expansions of capability include a mini-computer operating system, and advanced peripherals.

In 1974, a project to design a PLATO IV terminal which would combine low cost with expanded terminal capabilities was started under the supervision of J. E. Stifle. Some of the results of the earlier project have been included, and the finished design will be used as a prototype for the next generation of PLATO terminals (4). Several versions of this device, which is based on an INTEL 8080 and a parallel plasma panel, have been completed. The resident system, currently stored in re-programmable memory, includes block erase, double sized characters, programmable margins and tabs, and multi-directional text display. Some random access memory is available for user programs, which can be called from a PLATO lesson. Work is still being done to determine what other features should be part of the standard system and which should be offered as user programs.

4. ANALYSIS OF CURRENT TEXT TRANSMISSION METHODS

4.1 Background for Analysis

For a system such as PLATO IV with a large number of interactive terminals running simultaneously, host-to-terminal communication is a major part of the system load. With the design of the next generation terminal nearing completion, it seemed advantageous to study the overall system format from a communication/information point of view. First, it was necessary to determine the current distribution of display type information. From this distribution, it can be shown that text constitutes the major part of display activity. Therefore, ways to optimize the average number of bits/character sent has been the major emphasis of this project. Starting with a detailed analysis of the current character transmission method, both optimization of the current scheme and methods requiring more radical changes to the system will be discussed. Both character-by-character and word-by-word compression methods have been considered. However, it has been assumed that no basic changes to the overall communications hardware will be made.

One way of determining the distribution by display type of the information sent to the terminals is to monitor the output of the Frameater or of the PPU (Figure 2.3). At the time it was not practical to put a monitor at either location. The easiest place to sample was at the ECS resident system output buffer. The effect on the output stream could then be deduced. Using this method, one can determine that approximately 50% of all output is characters, 30% screen positioning information, and 20% lines. However, of the 30% screen positioning information, almost

25% of the 30% is taken up by returning to a software set margin. This will be eliminated by the variable set margins, already standard for the new terminals. It therefore seems most profitable to optimize text transmission. A description of the current character encoding methods for the terminal and the central system follows.

4.2 Terminal Character Format

The present PLATO IV terminal recognizes two types of 20 bit parcels or words: control and data. Normally, the Load Mode control word is used to set the terminal mode to either line, character, dot, or load user character memory. All data words that follow are interpreted relative to the mode. Control words include load mode, set x/y, and references to external devices.

The character format for the terminal involves the use of 6 bits packed three to a 20 bit data word. Bit 19 = 1 indicates that the word contains an 18 bit field of data. (Figure 4.1)

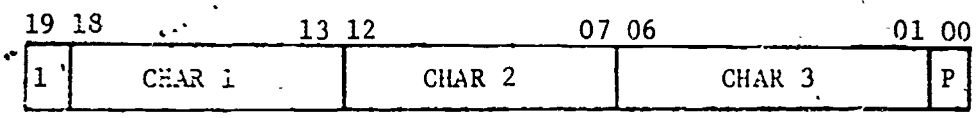


Figure 4.1. Character Mode Data Word

The 252 possible characters are arranged in 4 memories of 64 characters each. One character position in each memory (o77, where the preceding "o" indicates an octal number) is defined as an "uncover" code. The combination of an uncover code and another 6 bit code is used to indicate a change into another memory, or one of several special functions as described in Table 4.1.

To plot characters, the terminal is first set into character mode with a load mode control word. All subsequent data words are interpreted as above. Each character plotted automatically increments x by 8. Note that the carriage return function (o7715) is only useful in the special case where the left margin is at x = 0. To set either x or y, a 20 bit control word must be sent to the terminal. However, this is done without affecting the terminal mode.

4.3 Central System Character Format

Within the central computer system, characters are also kept as 6 bit codes. Since there are 252 characters plus special functions, combinations of 6 bit codes are necessary. The combinations are rather complex. The code o75, called font, is used as a locking toggle to delineate the alternate font, that is, the user programmable character memory. Within the set of 126 characters of either font, two more special codes are used; shift (o70) and access (o76). The following combinations are possible: 6 bit code; shift + 6 bit code; access + 6 bit code; access + shift + 6 bit code. Therefore, a maximum of 18 bits can be used to designate a character in either font. Other special codes are used to indicate positioning information such as superscript, subscript, etc. A complete list is given in Table 4.2.

This rather awkward encoding scheme is much more consistent when thought of relative to the key presses needed to create particular characters. The shift code directly relates to the upper and lower case "shift key" on

Table 4.1 Control Functions Following an Uncover (o77) Code

Code	Name	Function
o00	character NOP	no change
o10	backspace	x ← x-8
o11	tab	x ← x+8
o12	line feed	y ← y-16
o14	form feed	x ← 0, y ← 496
o15	carriage return	x ← 0, y ← y-16
o16	superscript	y ← y+5
o17	subscript	y ← y+5
o20	select M0	set to character memory 0
o21	select M1	set to character memory 1
o22	select M2	set to character memory 2
o23	select M3	set to character memory 3

Table 4.2 S cial Function Codes for Central Computer Encoding Scheme

Code	Name	Function	Terminal Code
o66	subscript	non-locking $y \leftarrow y-5$ for 1 character then y restored	o77 17, after the character, send o77 16 (unlock)
o67	superscript	non-locking $y \leftarrow y+5$ for 1 character, then y restored	o77 16, after the character, send o77 17 (unlock)
o70	shift	character definition	approximately selects M1 not complete correspondence
o71	margin return (carriage return)	$x \leftarrow 0$ $y \leftarrow y-16$	o77 15
o74	backspace	$x \leftarrow x-8$	o77 10
o75	font	define alternate font	following characters will be in M3 or M4
o76	access	character definition	approximately selects M1 not complete correspondence
o7066	locking subscript	$y \leftarrow y-5$	o77 17
o7067	locking superscript	$y \leftarrow y+5$	o77 16

a typewriter style keyboard. The characters preceded by an access are not visible on the key caps and are mostly mathematical or foreign language symbols. Effort has been made to relate the key to the symbol, such as defining π as access p. While this is the historical basis for the coding scheme, it is not necessary to keep it this way. The elimination of the 18 bit access-shift-character combination would considerably simplify character string manipulation, including the translation to output format. No additional overhead would be involved storing input keys, since, for most cases, a translation is already made between the value produced by the keyset and the value described above.

4.4 Description of Text Transmission

Using a 6 bit code for transmission to the terminal has two major advantages. First, 6 bits per character will fit into 18 bits with no overhead. Second, it is possible that an average of less than 8 bits/character, which is the number needed for a straight binary coding method, can be obtained because there should be relatively little switching between terminal memories. While certain foreign language and scientific symbols must readily be available in an education-oriented system, it is not expected that the average frequency of these symbols will be very high. Therefore, it should be possible to optimize the character transmission r te by carefully distributing the characters among the memories. This can be done by grouping all frequently used cha ters together, although what symbols are used in combinations must also be considered. It was decided to place the lower case alphanumerics plus commonly used punctuation and arithmetic symbols together in MO as letters and numbers are commonly found together when editing program text. All other ROM characters are in M1. These groupings can be seen in Figure 4.2. It was expected that foreign language lessons using a non-Roman alphabet would arrange the characters similarly in M2 and M3.

The following discussion will be based on the results of a system-wide sampling program. Details on this program can be found in Appendix A.2. These particular numbers are taken from an approximately one million character sample taken periodically throughout one afternoon. Although one million characters accounts for less than ten minutes of the total output flow from PLATO IV at such a time, the distributed sampling technique should give an accurate picture of the average situation. While a rigorous analysis has not been done to prove that this is true, several such samples have been taken and are consistent.

The actual character distribution can be seen in Figures 4.3 and 4.4. The space code is by far the most frequent character. In this sample, it represents around 25% of all characters sent, while 20% is considered typical for English text. The difference is partially due to the lack of a multi-character TAB function which requires that space strings be sent instead. Note that the space character appears both in M0 and M1, to avoid memory switching for this common case. After the space, the lower case alphabetic characters follow the normal English distribution.

In this particular sample, several character codes do not appear at all. One of these, the arrow seen at the far right in Figure 4.3, is actually quite prevalent system-wide. However, due to historical reasons, it is not encoded in the same manner as the other characters in the system output buffer, and as such was not seen by the sampling

ADDRESS (OCTAL)	MO CHAR	M1 CHAR	ADDRESS (OCTAL)	MO CHAR	M1 CHAR
0	:	#	40	5	↑
1	a	A	41	6	→
2	b	B	42	7	↓
3	c	C	43	8	←
4	d	D	44	9	∿
5	e	E	45	+	Σ
6	f	F	46	-	Δ
7	g	G	47	*	∪
10	h	H	50	/	∩
11	i	I	51	(	{
12	j	J	52	)	}
13	k	K	53	$	&
14	l	L	54	=	≠
15	m	M	55	SP	SP
16	n	N	56	,	|
17	o	O	57	.	°
20	p	P	60	÷	≡
21	q	Q	61	[	α
22	r	R	62	]	β
23	s	S	63	%	δ
24	t	T	64	×	λ
25	u	U	65	⇦	μ
26	v	V	66	'	π
27	w	W	67	"	ρ
30	x	X	70	!	σ
31	y	Y	71	;	ω
32	z	Z	72	<	≤
33	0	~	73	>	≥
34	1	¨	74	∸	θ
35	2	^	75	?	@
36	3	´	76	≫	\
37	4	`	77	UNCOVER	UNCOVER

Figure 4.2. ROM character memories.

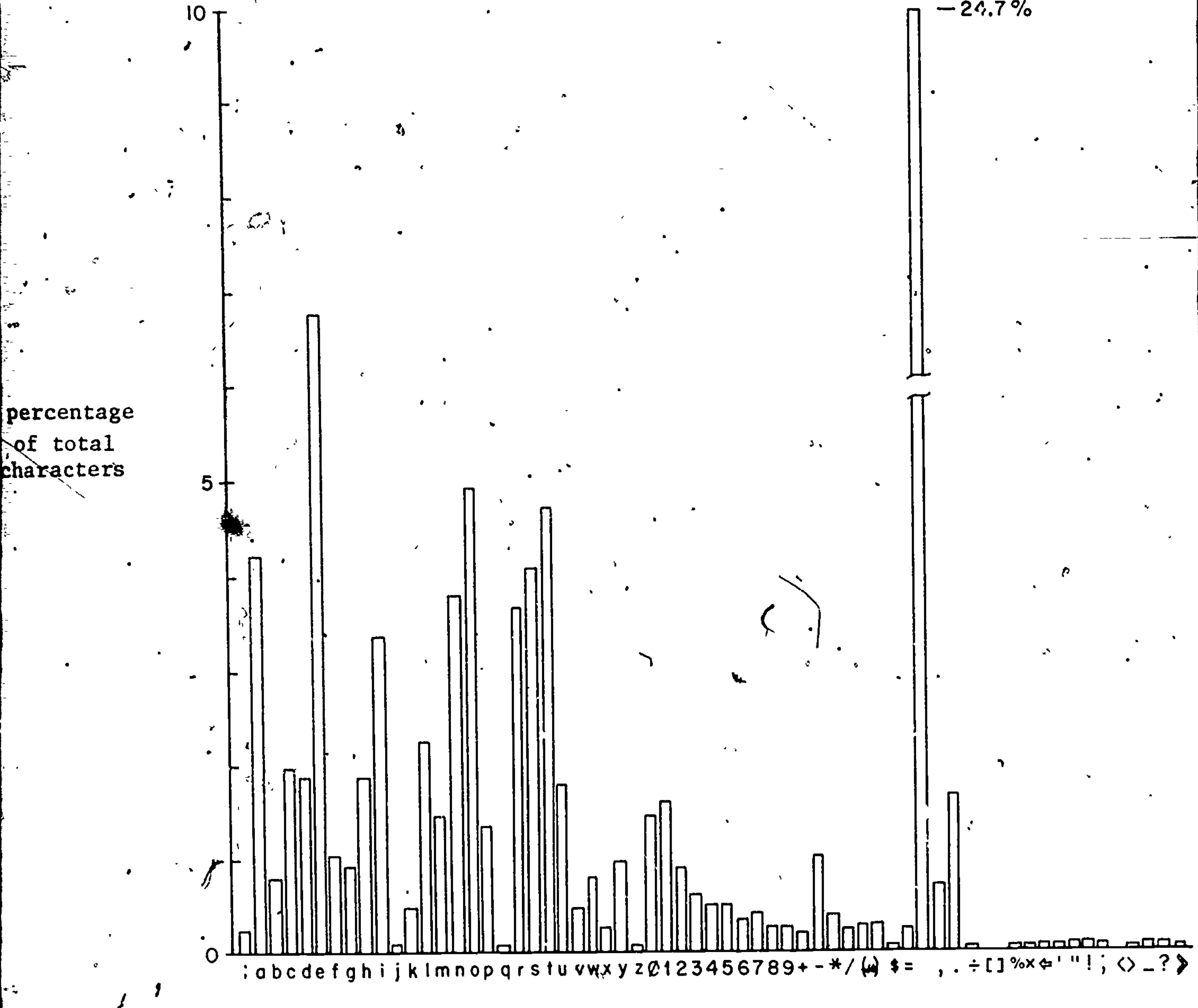


Figure 4.3. Character frequency distribution for MO.

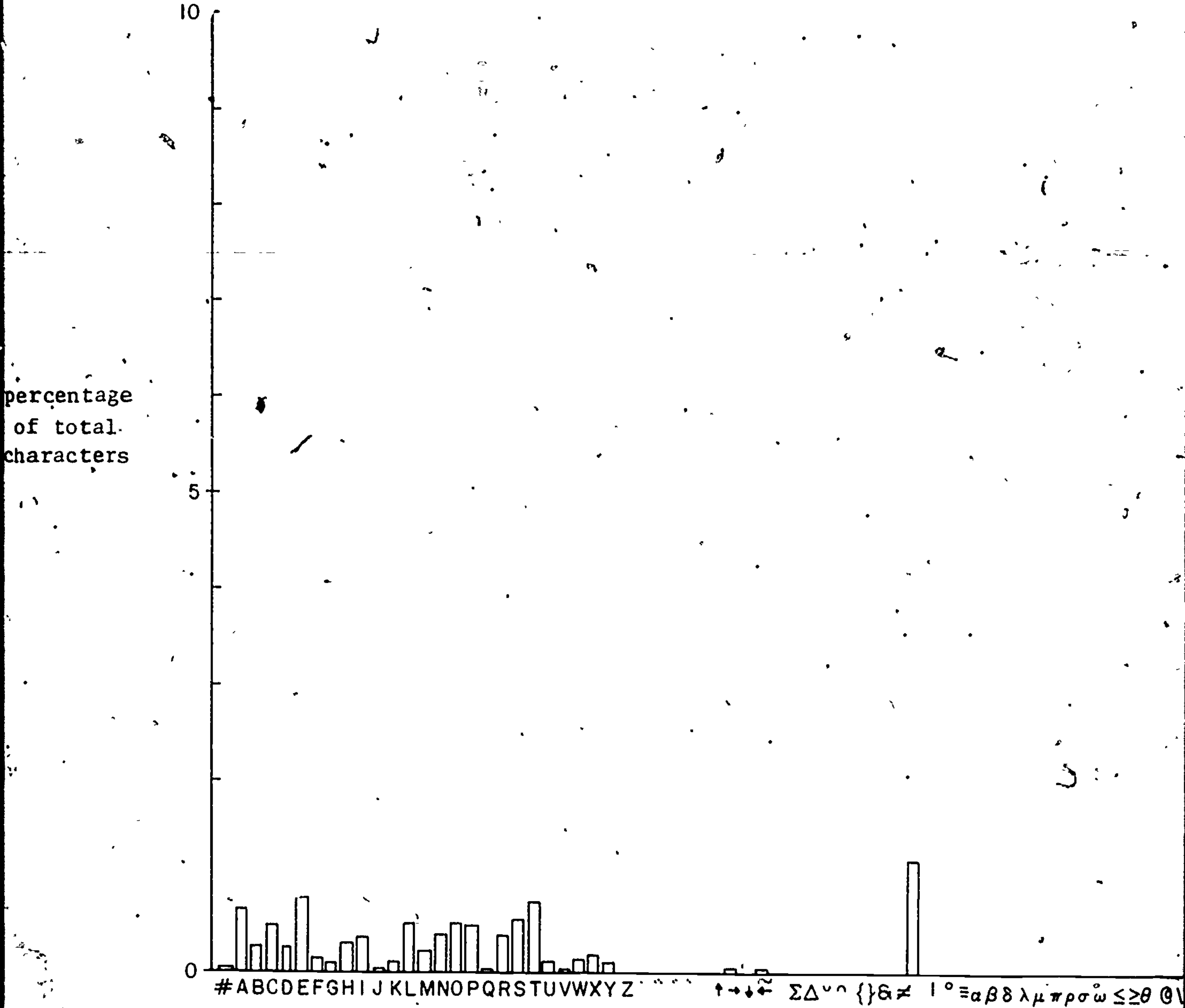


Figure 4.4. Character frequency distribution for M1.

program. Other characters that do not appear can be assumed to be infrequently used by the system as a whole. On inspection, it can be seen that they are either special mathematical symbols or foreign language symbols, which are very dependent on the type of lessons running. The type of lessons running depends on which classes are being taught at the time of the sample. These characters do appear in more selective samples.

Besides the character frequency data, information on individual memory usage and the distribution of memory transitions was taken over the same sample. The results indicate that 88.1% of all characters plotted resided in M0, 8.0% resided in M1, 2.9% in M2, and 0.9% in M3. As was anticipated, M0 is by far the most heavily used.

Inherent in this coding scheme is the assumption that once a change into a memory is made, the next character is more likely to be in the new memory than the old. This assumption can be checked by comparing the number of transitions out of a memory with the number of times the next character was within the same memory. In the case of M0, it is 20 times more likely that the next character is in M0 than in any of the other three memories. For M1, on the other hand, it is only 23% more likely that the next character is in M1 as opposed to anywhere else. Because M1 contains the upper case alphabet, it was suspected that the M0→M1→M0 transition, which would occur for a word beginning with a capital letter, would be quite frequent. Therefore, a special check for this transition was included. It was found that approximately 60% of the transitions between M0 and M1 were encompassed by this case.

This implies that a non-locking shift to M1 in addition to the current locking transition would be beneficial.

From the same data, it can be determined that 90.5% of the time, plotting a character does not require a change of terminal memory. This can be used to compute the average number of bits/character as follows:

$$.905 \times 6 + .095 \times 18 = 7.14 \text{ bits/character}$$

This is indeed better than 8 bits/character, as was predicted. This is not a completely accurate picture, however. Because of the overhead inherent in the 20 bit parcel scheme, the real number is somewhat higher.

First, each character actually requires 6.3 bits, to include the data/control bit. Recomputing gives 7.47 bits/character. Neither the start nor the parity bits are represented as they are not usually included in a discussion of this kind. However, the effects of these bits would be computed similarly. For ease of discussion, a 6 bit character will be assumed for the rest of this chapter unless explicitly stated otherwise. The higher value can always be obtained by multiplying by 6.3/6.0.

Another source of overhead is due to the fact that there are multiple characters in one data word. This can cause unused bits at the end of a character string. Within the current design, there is no 6 bit code which can be used as a NOP, or fill character. Therefore, it is necessary to go to a 12 bit NOP. The extra bits transmitted in this manner account for 12% of all character output. This increases the average number of bits per character to 8.00. This is the number of bits required by a straight binary encoding scheme, although it would not be possible to implement such a scheme directly without considerable overhead if the 20-bit parcel size were retained. It seems safe to assume that the use of a 6 bit NOP would reduce the fill overhead to 6%. A 6% overhead gives 7.57 bits/character.

Ignoring the 12% fill overhead for the moment, the result of translating this sampling of the output buffer to the format required by the terminal gives 7.64 as the average number of bits per visible character. The difference between this figure and the 7.14 bits/character given before is due to the function codes included in the output stream. Function codes are those codes described in Table 4.1, other than those used to change memories. Each code is assumed to take 12 bits. A discussion of the effect of the various types of function codes follows.

The most common single code is the margin return, or carriage return. Alone, it accounts for 0.4% of the character output streams. Because the new terminals will have programmable margins, it is expected that this function will become even more significant.

Taken together, the superscript, subscript, locking superscript, and locking subscript constitute 1.1% of the total character output. While the locking type can be sent with a 12 bit code, to do a non-locking superscript or subscript requires 24 bits. For example, to do a non-locking superscript requires a 12 bit locking superscript code to precede the character, and a 12 bit locking subscript code to follow it. In this sample, the extra overhead caused by not having a 12 bit unlocking superscript and subscript accounts for 0.4% of the total character output stream. While this number is not very large, for certain types of displays the overhead can be significant. For example, take the equation: $y = x_1^2 + 2x_1x_2 + c_1$. There are 14 visible characters, but the superscripts and subscripts require transmitting 20 more. This decreases the character writing rate to approximately 1/3 of what would be predicted by the 14 visible characters alone. Just using a 12 bit code would double the display rate, which is a visible increase in speed. This type of equation is common in mathematical and scientific lessons. For example, a sample of chemistry lessons showed that the average overhead for superscripts and subscripts was 6%. Furthermore, the locking case was used hardly at all relative to the non-locking case. For the sake of these special cases, a non-locking superscript and subscript function should be considered.

The remaining function codes, with backspace predominant, account for 1.18% of the total character output stream. To summarize: the function codes, assumming 12 bits/code except for the non-locking superscript and subscript which are 24 bits long, are 2.68% of the character output stream. While this number is small, a page of text with a large number of these codes can plot significantly slower because of the relatively large overhead for the code.

4.5 Recommendations for Improvement

Three areas for possible improvement have been identified: the 6 bit as opposed to the 12 bit NOP or fill characters, the non-locking transition from M0 to M1, and the non-locking superscript and subscript.

Below is a description of the effect on the average number of bits/character for each of these. For the rest of this discussion, the value computed using 6.3 bits/character to include the data/control bit, will be given in parentheses next to the value using 6 bits/character.

The base figure for comparison is the current average bits/character as computed by the following expression:

$1.12 \times 6(v + 2(t + f) + 2(usub + usup))/v = 8.55\ (9.0)$ bits/visible character,

where:

v = number of visible characters in the sample;

t = number of memory transitions in the sample;

f = number of function codes in the sample;

usub = number of unlocking subscripts in the sample;

usup = number of unlocking superscripts in the sample.

Reducing the fill overhead to 6% gives 8.10 (8.5) bits/character.

Using a 12 bit, non-locking transition for $M0 \rightarrow M1 \rightarrow M0$, but still assuming 12% fill gives 8.40 (8.84) bits/character. With 6% fill, it reduces to 7.96 (8.36) bits/character.

Changing only the non-locking superscript and subscript transmission gives a value of 8.52 (8.95) bits/character. As discussed previously, the effect of this on the average is slight.

Implementing all three optimizations gives 8.06 (8.50) bits/character. This is an overall savings of ½ bit per character. While this is only a 5.6% increase in display speed, none of these improvements should be particularly difficult to implement. As was previously pointed out, using a non-locking superscript and subscript could give a visible speed increase in some situations. The 6 bit NOP would require the loss of a character code. However, the eliminated character could be retained through a 12 bit control function, or the number of memories could be expanded. How many characters can be stored will eventually be limited by the cost of the hardware.

5. VARIABLE LENGTH CODING

5.1 Introduction and Description of Basic Principles

The previous chapter has given an analysis of the current status of character transmission in PLATO IV, and listed three areas of possible improvement. All together, the average increase in transmission rate would be only 6.0%, however. To obtain a more significant increase in transmission rate, and thus display speed, it is necessary to look at more sophisticated methods of compression. In this chapter, a definition of variable length or Huffman coding will be presented, followed by a discussion of its applicability to the PLATO IV system. The basic assumption will be that the communications hardware will remain unchanged. That is, transmission will occur synchronously, in 21-bit parcels, 18 bits of which can be character data, and that transmission speed will be limited to 1200 baud by the voice grade phone line.

Within any transmission scheme, there is a finite set of symbols that represent all possible messages sent by the system. The information content for a particular symbol is a function not only of the total number of possible messages, but of the probability of occurrence of the symbol itself. An "optimal" encoding scheme is one which transmits no redundant information. To create an optimal code, it is necessary to have the number of bits used by a particular symbol be inversely proportional to the frequency of the symbol. In comparison, most computer character codes use a fixed number of bits/character, determined by the number of different characters. This method would only be optimal if all characters were equally likely, which is obviously not the case.

A method for creating minimum redundancy, or optimal codes from a set of symbols and their relative frequencies was described by Huffman in 1952 (7). These codes have the following properties: 1) The codewords have lengths inversely proportional to their frequencies. That is, the most frequent codewords are the shortest ones. 2) Codewords are assign to the bit patterns such that there are no unused sequences shorter than the longest codeword. 3) No valid codeword begins with a shorter valid codeword. Therefore, there is no need to include any extra bits to define the start or end of a codeword. The shortest valid sequence is guaranteed to be the correct one.

Figure 5.1 gives a brief example of such a code. For a description of how to derive such a code, the reader is referred to Huffman's article (7). In this example, there are five possible messages. If a fixed length code were used, three bits/message would be required. Using the Huffman algorithm to define the number of bits for each message, the average can be reduced to 1.9 bits/message. One possible set of codewords has been assigned.

It is possible to determine the optimal number of bits needed to transmit the information from the relative frequencies of a set of symbols without actually constructing the minimum redundancy code. The formula is:

average bits/character = H/total # characters in sample

where H is the entropy function defined by:

i	P(i)	L(i)	P(i)L(i)	codeword
1	0.50	1	0.5	1
2	0.20	2	0.4	01
3	0.20	3	0.6	001
4	0.08	4	0.32	0001
5	0.02	4	0.08	0000
			$1.9 = L_{av}$	

Figure 5.1 Where i = the message number; P(i) = probability of occurrence of message number i; L(i) = length of the codeword for i: codeword = bit pattern for i. The sum of P(i)L(i) for all i gives the average number of bits/message.

$$H = \sum_i \frac{f_i}{f_{total}} \log_2\left(\frac{f_{total}}{f_i}\right) \quad \text{for all } i \in \text{sample}$$

f_i = frequency of i^{th} element

$f_{total} = \sum_i f_i$ = total characters in sample

For the sample used in the previous chapter, this gives 4.95 bits/character. This is 33% shorter than the 7.5 bits/character currently available as the theoretical limit to the PLATO IV coding scheme.

5.2 Implementation on PLATO IV

The implementation of a variable length code on a system like PLATO IV could be done as follows. To encode, a table lookup can be used. This is already done for the current encoding scheme. The characters are then packed into the 18 data bits and transmitted. A fill pattern, such as all 1's, would be used only at the end of text transmission, since character codes can be decoded even if they overlap parcel boundaries.

To decode a variable length code, it is only necessary to consider the character input as a stream of bit . Each bit is examined in turn until a codeword is found. This can then be decoded and the next character started. Since this is a serial operation, it is not necessary to have an integer number of character codes within a parcel. The decoding algorithm can be likened to moving along a binary tree, where each bit determines either a left or right branch. When a leaf is reached, the codeword has been found.

For any new character coding scheme on PLATO IV, care must be taken to include the function codes in the set of transmission symbols. While it is common terminology to refer to the number of characters as 256 (or 252), this is not the case. The actual figure that should be used is 265 for the current system (252 + uncover + 12 functions) and at least 274 for the projected terminal [4].

6. THE USE OF WORD LISTS OR DICTIONARIES

6.1 Introduction to Dictionary Compression Methods

Up to this point, transmission of text has only been discussed in terms of transmission of a string of character codes. However, the amount of information available in a page of text is not defined only by the information inherent in the individual characters. The organization of these characters into words is also significant. Including this information in a text encoding scheme can be used to drastically reduce the average number of bits per character required. The theoretical limit, as defined experimentally by Shannon in 1951, is 1.3 bits/character (6). Algorithms as efficient as 1.8 bits/character have been defined for computer systems, using dictionaries of words and word by word encoding (8).

The method used is to create a word list or dictionary containing some or all of the words in the text. Each word in the dictionary is assigned an index indicating its position in the list. To encode, this index is substituted for the word in the text. Traditionally, this method has been used to decrease storage requirements, especially for archival storage because to obtain maximum compression requires the use of large dictionaries. Therefore, encoding time, which requires a search through the word list, can be high. However, a study made by Godfred Dewey (9) of printed text indicates that the word "the" alone accounts for more than 7% of all printed text. He also indicates that the first 10 words by frequency account for more than 25%, and the first

100 words account for more than 50% of all printed text. Therefore, it would seem that a significant benefit could be obtained by using a relatively short list of words.

To use dictionaries for host-to-terminal transmission, three areas have to be considered: the distribution of words transmitted by the system, since it is not guaranteed to be the same as that for printed English; the ability of the terminal to decode and plot the received word; and the amount of extra overhead at the central computer caused by the encoding.

6.2 Word Distribution on PLATO IV

To study the word frequency distribution, the program which takes periodic samples from the system output buffer as described in Chapter 4 was used. The sample was then parsed into words and a frequency count for each word was kept. From this list, the impact of dictionaries, on the average, could be deduced. In this program, while the space code was included as a delimiter, some samples were analyzed which also counted space strings as words to predict the benefits of the programmable tab. Further details on the mechanics of this program can be found in section A.3 of the appendix.

The results of this program show that while the frequency distribution is similar to that given for English (9), many of the more frequent words are peculiar to PLATO IV. Notably, words indicating keys to be pressed, plus the word "press" itself were very common. For one sample of approximately 100,000 words, not including space strings, the most common word was "the", which was 4.6% of all words transmitted. The first 10 most frequent words include 16.7%, and the first 100 words include 44.3% of all words transmitted. A similar sample, including space strings, gives the double space as the most frequent, at 7.9%, followed by "the" at 2.75%. The first 10 words give 22.2%, and the first 100, 46.0% of all transmitted words.

While the above numbers offer the most direct comparison of PLATO word distribution with other word frequency studies, to determine the effect of a dictionary encoding scheme on transmission speed it is necessary to look at a slightly different measurement. What is needed is the amount of the total output flow that is described by the words. This number is computed as follows:

length x frequency / total characters

length = # characters needed to transmit the word

frequency = frequency of occurrence

total characters = total number of characters, including delimiters, transmitted for the entire sample

It was assumed that a space code would be transmitted with the word except in the case of the space strings.

For the sample without the space strings, transmitting the most frequent word, "the", plus a space defined 3.9% of the total character output. The first 10 words encompassed 14.5%, and the first 100, 38.0% of the transmitted characters. For the sample with the space strings, the results were 8.3% for the first word (double space), 23.4% for the first 10, and 47% for the first 100 words.

6.3 Decoding Algorithms

To decode a dictionary encoded text, it is necessary to know the dictionary, and, if not every word in the text is in the dictionary, to be able to distinguish character codes from word indexes. A simple method compatible with the current method of transmitting characters on PLATO IV would be to have memories similar to M0 and M1, which contain whole words as entries. Words in the "word memories" would then be accessed by selecting the memory with an uncover code, then sending a 6 bit index to select the word. Statistics could be taken to determine whether a locking or unlocking selection would be more efficient. This algorithm, using unlocking transitions, was implemented on the PDP-11 based programmable terminal, and was used to display a sample text with a 30% increase in speed. Unfortunately, to achieve any gains, the words in the memories have to have a transmitted length of greater than 3 characters, as it takes three 6 bit codes to select the word. Most common words are short, so savings obtained by this method would not be very great.

A more efficient variation of this method interleaves characters and words in the same memories. The more common words occur more often than many characters, so the optimal method would be to place the most common words in M0, moving some of the less common letters and symbols in M1. M1 would also contain words as well as letters. The number of new memories needed would then be a function of the number of words added.

Internal to the terminal, the memories would not need to be physically interleaved. Then, however, a translation table would be necessary. This sort of logic could easily be handled by a micro-processor.

Assuming absolute best case, that is, that it takes no more than 6 bits to access a word, the following savings could be obtained. Including space strings, a 10% reduction in output could be obtained with 15 words, a 20% reduction with 52 words, and a 30% reduction with 100 words. Not including space strings requires 26 words for a 10% reduction, 70 words for a 20% reduction, and 130 words for a 30% reduction in text output. The figures were obtained using a formula similar to the previous one:

(length - 1)(frequency) ' total characters

where the -1 indicates the 6 bits/word needed for transmission.

The previous discussion assumed that the same word list was used for all students. However, the words that are universally common are also short. If the vocabulary were tailored to the lesson, longer words could possibly result in higher savings.

A sample taken from students running organic chemistry lessons was analyzed. The results showed that while the word distribution was distinctly oriented towards organic chemistry, the percent of the characters encompassed by the most frequent words was only slightly higher than for the more general case. For the most common word, CH, the percent savings was 2.19. For the first 10 words, the savings was 10%, and for the first 100, it was 34.4%.

Another specific sample was taken from the system editor. Since the language being displayed is fixed format, the space strings used as tabs were most predominant, followed by those words in the heading for each page. The first 10 words give 19.7% of the characters. However, 7 out of the first 10 words are space strings, which could be replaced by a tab function.

There is the additional problem with programmable dictionaries of loading the dictionary. However, this could be accomplished in the same way as loading the programmable characters set. The average number of 6 bit characters per word is around 6.5. Assuming 3 characters every 1/60 of a second, a 100 word dictionary would take less than 5 seconds to load. Up to 17 seconds is needed to load the programmable character set, so a 5 second wait would not be unreasonable.

6.4 Cost of the Encoding Method

It has been shown that approximately a 30% decrease in the information flow, which would correspond to a 43% increase in display speed, could be obtained using a 100 word dictionary. It is also well within the capabilities of the terminal to decode the information. We must now examine the cost of encoding such a scheme.

The optimal place to encode is in the Frameater, since the text string is already being encoded there. The additional overhead for word by word encoding would be the time needed to parse the word, the table storage space, and the time needed for the table lookup. The overhead involved with the table lookup is not excessive. Likewise, for a fixed table for all users, the storage requirement is trivial. However, if user defined tables are used, a separate table for each user must be stored. For a system that runs over 400 terminals simultaneously, this overhead can be significant, especially since the tables would have to be kept in ECS.

The amount of CPU power that is currently used in formating is conservatively estimated as 1/3 of all PLATO operations. Of this time, the largest part is spent formating text not only because text is the major portion of the output flow, but because the formating process for text is relatively time consuming. Parsing for words would add the overhead of searching for delimiters to each character processed. Under current conditions, the increase in processing time caused by this procedure would degrade system performance enough to completely nullify any gains in display speed obtained b. using dictionary encoding.

7. CONCLUSIONS AND FUTURE PROJECTS

7.1 Summary of Results

In this paper, an attempt has been made to show how one might increase the speed of character displays on PLATO IV, or a similar system. First, the currently used method was analyzed, and an average rate of 9.0 bits/character was computed for a typical sample. Three areas of improvement were defined which would decrease the bits/character to 8.5, a change of 6%. This implies only a 5.6% increase in display speed.

Second, the limit obtainable using Huffman coding was computed to be 4.95 bits/character for the same sample. As this is calculated without including the overhead generated by end of text fill, or the data/control bit, it is necessary to compare it to 7.5 bits/character, which is the equivalent figure for the optimized version of the current method. This implies an increase in display speed of 50%, or 1-1/2 times faster.

Chapter 6 discussed word list encoding. Using approximately the same type of 6 bit code as is now used to encode characters to encode words, a 30% decrease in the volume of text information could be obtained using a 100 word dictionary. This would give a 43% increase in display speed. However, the overhead to encode the words is prohibitive, even for short lists.

In summary, while some special cases can be improved by modifying the currently used method for text transmission, a completely new coding scheme must be constructed to achieve any significant increase in average transmission rate. Using a variable length code will give a minimum increase of 50% over current display speeds. However, it is unlikely that such a code will do more than double the display rate.

It is possible to work with a combination of word lists and Huffman coding to obtain greater compression. One possible algorithm for this is outlined below. However, for many cases it is not the average rate which is most significant in terms of display esthetics, but the "burst" rate. For example, it often occurs that a complicated display will be transmitted to a terminal, then transmission will stop, or be reduced to a very low level while the user studies the display. Therefore, the average rate of transmission is low, but esthetically the process is slow because of the large amount of time needed to plot the display. Sub equent replots of the display are even more tedious. Suggestions for improving burst display speed for some cases are given in Section 7.3.

7.2 Suggestions for Future Work in Text Compression

To obtain greater increases than the 50% mentioned above, it would be necessary to go to a combination of methods, such as using Huffman coding with word dictionaries. While this retains the problems of processing overhead, a variation of this might be possible. It was mentioned in Chapter 6 that the double space was a very common pattern. Other two-character combinations, which were not analyzed as they were not classified as words by the program, are also common: A coding algorithm using only 1 and 2 character groups would be less expensive than the dictionary lookup, since the Frameater would not have to search for delimiters. A modified indexing scheme could be used to reduce the search time for valid double character groups. For example, the first character would be used as an index, as it is now, into an encoding table. Each table entry could contain a pointer to a list of double character groups beginning with that character. Thus, a very short table lookup would be the only major overhead. The program which now takes statistics on word frequencies could easily be modified to study this and other multi-character groups.

7.3 Increasing "Burst" Display Speeds

Some experimentation has shown that an increase of average display rate of 20% relative to the current rate of approximately 120 characters/second is scarcely visible. Doubling the rate to 240 characters/second begins to give significant advantages for full screen displays. However, the maximum rate for the parallel plasma panel is nearly 6000 characters/second. At that rate, it takes only 1/3 of a second to fill the screen. There is no way to use that ability by relying strictly on the average data rate over a 1200 baud line. Even considering the limitations of the 8 bit micro-processor and using 2000 characters/second as a maximum, this is an order of magnitude more than what was predicted for any of the general text encoding methods. However, it should be possible to use the high speed display in bursts.

One example of such a burst operation is block erase. There, it takes relatively little information sent from the central computer to indicate the rectangular area. Then the local processor can erase the area at as high a speed as possible, limited only by the local processor and the display. The same principle as block erase can be used for area shading.

This burst capability can be extended to text by storing locally common headings, help sequences, or index pages in a manner similar to the image trapping mentioned in Chapter 3. Also, the user programmable character set is often used to make small, multi-character pictures. After a certain size, it is possible to see the individual characters within the pictures plot. If a translation table were stored locally, indicating which characters fit together, then each figure could be called by a single character code transmitted from the main computer. Especially for characters involved in animations, the improvement in display quality would be considerable.

Another area that can be greatly improved in a burst mode is line drawing. The current method sends an endpoint every 17 msec. For a complicated figure, it may take ½ minute to plot. There are several ways to improve this for special cases. First, it is possible to use image trapping. Second, many line drawings are actually sized characters. Moving the ability to compress and expand character size to the terminal, if possible, would significantly increase the speed of such displays. Other than that, it is necessary to find some method of packing more endpoints in 18 bits of data.

The resolution of the plasma display is 512 x 512, 60 lines/inch. Therefore, it takes 9 bits to give maximum x or y, and 6 bits to describe an inch. One possibility is to pack Δx, Δy, and try to get three coordinates into 18 bits. As in character strings, it is not essential that whole endpoints arrive in one parcel. However, the decoding operation is not as convenient for such a case here.

Another possibility is to define a larger grid for lines, so that it takes less bits for maximum x and y. Six bit resolution gives a grid of approximately 1/8 of an inch. In fact, there is a commonly used coarse grid already on PLATO IV, corresponding to the character grid, which is 8 x 16 dots. This grid is often also used for lines as well.

A special case can be made for horizontal and vertical lines, such that only one y or x coordinate, respectively, need be indicated. To determine which method would give the greatest gain, it would be necessary to do a sample and analysis program for lines, similar to the one done for characters. An attempt was made to use a modification of the character analysis program to study lines. However, the critical information for line is the distance between endpoints. A strict average would not give the information needed. Therefore, it would be necessary to keep more information as to where the lines are sent to guarantee valid results.

7.4 Elimination of Text Formating

It has been mentioned in Section 6.4 that approximately 1/3 of PLATO's CPU needs are required for formating. With a processor based terminal, it is possible to eliminate the character formating altogether by accepting the internal codes described in Section 4.3. As the system gets more processor bound, this becomes an increasingly attractive option. A program to do this has been written for the micro-processor based terminal, which is basically just a sparse table indexing routine. (12) While a full scale analysis of the internal codes with regards to transmission has not been done, it could easily be performed by modifying the character by character analysis program. Two things would be obvious improvements. First, eliminate the access + shift + 6 bit code characters. This would decrease the decoding table size by 25%. Second, add a lock shift. The relative merits of (approximately) shift and lock shift were discussed in 4.4 with regards to the M0→M1→M0 transition. It was found there that approximately 60% of all shifts are non-locking.

REFERENCES

1. B. Sherwood and J. Stifle, "The PLATO IV Communications System," CERL Report X-44, Computer-based Education Research Laboratory, University of Illinois (1975).

2. J. Stifle, "The PLATO IV Student Terminal," CERL Report X-15, Computer-based Education Research Laboratory, University of Illinois.

3. M. Stone, R. Bloemer, R. Feretich, and R. L. Johnson, "An Intelligent Graphics Terminal with Multi-Host System Compatability," Digest of Papers, CompCon Fall 74, pp. 37-40.

4. J. Stifle, "A Preliminary Report on the PLATO V Terminal," Internal report, Computer-based Education Research Laboratory, University of Illinois, May 19, 1975.

5. S. Smith and B. Sherwood, "Educational Uses of the PLATO Computers System," Science, Vol. 192, p. 344 (1976).

6. C. E. Shannon, "Prediction and Entropy of Printed English," Bell System Tech. J., 30, 50-64 (1951).

7. D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," Proc. IRE 40, 1098-1101 (1952).

8. R. D. Cullum, "A Method for the Removal of Redundancy in Printed Text," Thesis, University of Illinois Dept. of Computer Science (1972).

9. D. Godfrey, "Relative Frequency of English Speech Sounds," 1923, Harvard University Press, Cambridge, Massachusetts.

10. D. Bitzer, B. Sherwood, and P. Tenczar, "Computer-based Science Education," CERL Report X-37 (1972), Computer-based Education Research Laboratory, University of Illinois, reprinted in "New Trends in the Utilization of Educational Technology for Science Education," UNESCO, Paris (1974).

11. R. L. Johnson, private communication.

12. B. Sherwood, private communication.

APPENDIX

A.1 Sampling Program

This program periodically samples the system output buffer, screens the information, and places it in a disk file, called a dataset. The parameters for the screening process are: user type, course, lesson, station, and output header code. These items are described below.

There are two main user types, author and student. An author is assumed to be developing lesson material, while a student is studying it. Therefore, the author is often using the editor and other system utilities, while the student will e running under a specific set of lessons. The current average system load is approximately ½ students, and the number is increasing.

Each user is registered in a course. Especially for students, the general area of interest for the user can be determined from this course. For example, students in course chem136a are studying organic chemistry.

The lesson name can be used to define a very specific area of interest, such as the system editor. The station number, which defines a particular terminal, can be used to determine what output is sent to one user, or group of users such as the classroom at the Foreign Language Building.

The format for the system output buffer is a heading, followed by data, repeated. Included in the heading is a code to indicate how the data is to be interpreted. This code is called the output header code, and is used to distinguish characters from other types of output.

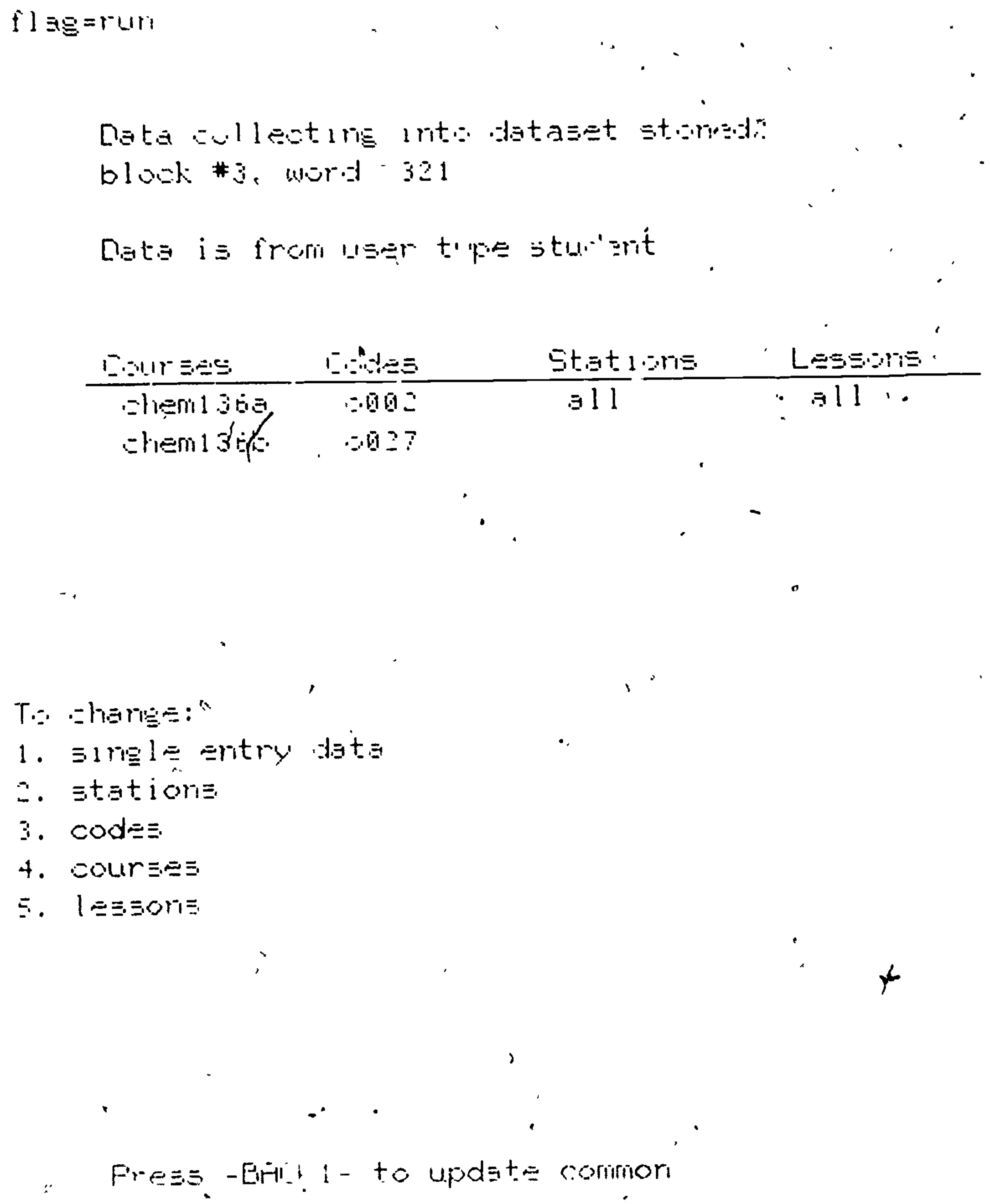
The screening parameters are kept in a table which can be edited by a separate program. A sample output, showing data being collected for all chemistry students enrolled in several sections of an organic chemistry curriculum, is given in Figure A.1. Output header codes o002 and o027 indicate text information. This same program can also be used to determine the amount of data sampled as there are five different datasets used to hold samples, each with 126 blocks of 322 words each.

The sampling program is automatically run every hour for a maximum of 10 minutes throughout the day.

A.2 Character-by-character Analysis Program

This program takes the character data stored by the dataset in the sampling program and produces the statistics discussed in Chapters 4 and 5. That is, it is used to determine the character frequency distribution, memory usage and memory transition information, and the data needed to compute the average bits per character sent under various conditions. A page of sample output for all but the character distribution is given in Figure A.2. A brief definition of each term on this page follows. Starting on the left:

PLATO characters:	the number of 6 bit internal codes processed for the sample
formatted characters:	the number of 6 bit codes sent to the terminal, not including fill
visible characters:	number of characters actually displayed. This is the same as summing the frequency distribution for all four memories.



```
flag=run

     Data collecting into dataset stored2
     block #3, word 321

     Data is from user type student


     Courses       Codes        Stations      Lessons
      chem136a      c002         all           all
      chem136b      c027


To change:
1. single entry data
2. stations
3. codes
4. courses
5. lessons


     Press -BACK1- to update common
```

Figure A.1. Display showing screening parameters for sampling program. In this example, text data is being collected for all students in chem136a and chem136b.

```
plato chars=922075
formatted chars=944167
visible characters=741653
total num. of trans=70542
#of M0→M1→M0 trans=16100

M0  =  653674   88.137%
M1  =   59698    8.049%
M2  =   21434    2.890%
M3  =    6849    0.923%

total =  741655


#shift     =104141   11.300%
#access    = 11995    1.301%
#font      = 19217    2.084%
#lock sup=      71    0.008%
#lock sub=  [illegible]    0.005
#backspace  =  [illegible]   0.508
#subscript  =  [illegible]   0.136%
#superscript=  [illegible]   0.072
#margin return= 19711   2.138
```

```
#bits/char=7.64 (8.02)
plus 12% fill=8.55 (8.9[illegible]

limit by shannon's
bound= 4.96 bits/char


0→0   619263  83.497%
0→1    26636   3.591%
0→2     6052   0.816%
0→3     1727   0.233%
1→0    26739   3.605%
1→1    32883   4.434%
1→2       52   0.007%
1→3       21   0.003%
2→0     5908   0.797%
2→1       55   0.007%
2→2    14796   1.995%
2→3      674   0.091%
3→0     1754   0.236%
3→1      124   0.017%
3→2      534   0.072%
3→3     4427   0.597%

tot    741655
```

Figure A.2. Sample display for character-by-character analysis program.

total number of transitions:	This is the number of requests for memory transitions.
# of M0→M1→M0 transitions:	This is the number of occurences of a M0→M1→M0 transition.

The next 5 lines give the character usage among the four memories. Both the total number of characters and the percentage of the total visible characters for each memory is given.

The frequency of occur   the special codes (shift, access, etc.) is then listed along with the percentage relative to he number of internal PLATO characters.

At the top right:

# bits/character:	This is 6 bits times the number of formatted characters divided by the number of visible characters. The number in parenthesis includes the data/control bit.

This number plus 12% fill is given in the next line, in the same format.

The limit by Shannon's bound is calculated from the character distribution using the formula given in 5.1.

The remai er of the display gives the transition information. For example, the entry labeled 0→0 indicates that out of 741655 visible characters, 619263 were displayed from M0 without any transition. Therefore, 83.50% of the time, the base memory was M0, and the next character was also in M0. Also, summing the four entries which indicate that the final memory was M0 gives the total number of characters displayed from that memory, which matches the entry for M0 on the left side. This provides an internal consistency check.

This program was also used to provide the information for the character frequency graphs drawn in Figures 4.3 and 4.4. A variation of this program was used to determine which type of display information was predominant. Another variation was used to try to find what length lines are common; however, it was decided that the sampling technique destroys that information. If the sampling program were modified, analysis of lines would be possible.

Future uses of the character-by-character analysis are: studying the internal format with regard to transmitting internal codes directly to the terminal, and analyzing the effectiveness of any system change.

A.3 Word-by-word Analysis Program

This program provides the word frequency distribution information for Chapter 6 from the data generated by the sampling program. First, the text is scanned for delimiters, which are all non-alphabetic characters. Anything between delimiters is considered a word. The words are kept in a table in ECS, in alphabetical order, which is updated to a disk file periodically. Each time a word is found, a b[illegible] y ch p is used to find the word in the table. If it is not there, it is inserted in the proper position. Each table entry is two 60 bit words long. Up to 17 6 bit codes are stored per entry. The remaining bits are used for frequency informat .n.

While collecting words, the table is allowed to grow to 6601 entries. Then it is sorted by frequency and the amount representing 3/4 of the total words are retained. This is usually around 600 entries.

The table is then resorted alphabetically, and the processing continued. A typical sample represents approximately 100,000 words.

The following calculations are performed on the table: sort by frequency, percentage of total words for each word, percentage of total characters for each word, percent savings for each word, and a running total for each of these.

The most cumbersome part of this program is the enormous amount of time needed to create the original word frequency table. Running under low system load, this takes several hours real time, not necessarily consecutively. The table lookup is expensive because the entire table will not fit in central memory. The binary chop was selected because it is a fast search routine, and it could be performed without transferring the entire table into central memory. Future uses of this program would be to study character grouping different than words, such as dipthongs. However, to be truly useful, the word gathering part must be made faster. Writing it in Fortran would be one possibility.

A.4 Word-by-word Analysis of Source Files

As a preliminary study, a program written in Fortran was used to compile word frequencies from lesson source code. However, it was felt that this could not be representative as it did not include the effect of repeated displays. Also, it required guessing the lesson mix to simulate the system load. However, for specific areas, such as one group of students, a reasonable approximation of the word frequency order can be obtained by scanning the lessons that are included in their curriculum.