SAND99-2892
Unlimited Release
Printed December 1999

LFSRs Do Not Provide Compression

Philip L. Campbell
Secure Networks & Information Systems
Lyndon G. Pierson
Advanced Networking Integration
P.O. Box 5800
Sandia National Laboratories
Albuquerque, New Mexico 87175-0449
{plcampb, lgpiers}@sandia.gov

Abstract

We show that for general input sets linear feedback shift registers (LFSRs) do not provide compression comparable to current, standard algorithms, at least not on the current, standard input files. Rather, LFSRs provide performance on a par with simple, run-length encoding schemes. We exercised three different ways of using LFSRs on the Canterbury, Canterbury (large set), the Calgory Corpora, and on three, large graphics files of our own.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (703) 605-6000
Web site: http://www.ntis.gov/ordering.htm

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

1 Introduction

Massey, in his 1969 paper presenting the Berlekamp-Massey linear feedback shift-register synthesis algorithm, encouraged the investigation of LFSRs as compressors:

> There appears to be a number of interesting applications for the LFSR synthesis algorithm of Section III. The most obvious is that of finding a simple digital device to generate a prescribed binary sequence with useful properties in some application. Less obviously, the algorithm might be used as part of a source code, or data compressor, for a binary data source whose output contains considerable redundancy. For instance, the source digits could be processed by the algorithm in blocks of 127 digits. Each block could then be represented for transmission as a 7-bit block giving the length L of the shortest LFSR that generates the original sequence, followed by L bits to indicate the values of the tap connections and a further L bits giving the initial contents of the LFSR. Therefore, a total of 2L+7 bits would be transmitted in place of the original 127 bits. Such a data compression scheme could be expected to perform efficiently only when the underlying constraints producing the source redundancy were with high probability linear relations among the binary source digits. (from Massey [1])

This paper follows up on Massey's suggestion: we test the value of LFSRs for compression.

We describe LFSRs in Section 2, present the Berlekamp-Massey algorithm in Section 3, present the options for a compression method based on LFSRs in Section 4, and present the parameters and inputs we use in Section 5. We show our results in Section 6, and discuss related work in Section 7.

2 A Primer on Linear Feedback Shift Registers

An LFSR consists of n "cells," $n \geq 0$, consists of n "cells," $n \geq 0$, and k "taps," $0 \leq k \leq n$. The value in each cell is initialized to either 0 or 1. For example, the following

cell numbers →	0	1	2	3	4
cells →	0	1	1	0	0
taps →			*		*

represents an LFSR of length 5 (i.e., with 5 cells) and two taps. The taps are at cell numbers 2 and 4, where the leftmost cell is cell number 0. This LFSR generates a binary sequence that begins with 001101110, where the leftmost bit is the first bit generated. (Menezes et al., [2])

The algorithm presumes the Fibonacci configuration. That is, an LFSR generates a bit by proceeding through the following three steps:

1. The binary values in the cells corresponding to the taps are added up, mod 2, to produce a new binary value that we will call, for the nonce, the "shift-in bit."
2. The LFSR is shifted to the right. The bit shifted out is the generated bit.

3. The leftmost cell in the LFSR is set to the value of the shift-in bit.

Note that if there are zero taps, then the shift-in bit is always zero. If there is only one tap, then the shift-in bit is always the value of the cell corresponding to the tap. And if there are at least two taps, then the shift-in bit is the result of the EXCLUSIVE-OR function.

Given the generating steps above, note that

- binary sequences of the form 0*, where * denotes zero or more appearances of the preceding symbol (Kleene closure), can be generated by the degenerate LFSR of length zero;
- binary sequences of the form 1^+, where $^+$ denotes at least one appearance of the preceding symbol, can be generated by an LFSR of length one, the one cell initialized to 1, with one tap;
- binary sequences of the form 10* can be generated by an LFSR of length one, the one cell initialized to 1, with zero taps;
- binary sequences of the form 0^+1 require an LFSR that is the same length as the input binary sequence, with any arrangement of taps (i.e., no compression).

3 The Berlekamp-Massey Shift-Register Synthesis Algorithm

The Berlekamp-Massey linear feedback shift-register synthesis algorithm produces an LFSR that will generate a given input binary sequence and whose length is such that no shorter LFSR will generate the same sequence. A description of the algorithm and a proof of its correctness (Massey [1]), an analysis of its running time (Gustavson [3]), and the implementation in C that we have used in this research (Campbell [4]) can be found elsewhere, as can general information on LFSRs (Schneier [5]). The importance of the algorithm for the current work is that it provides an efficient method of finding an LFSR that generates a given input bit sequence.

4 Experimental Approach

The purpose of this research is to evaluate the use of LFSRs for compression. However, there are many ways to use LFSRs for this purpose, and there are many ways to re-arrange the input prior to processing, such as reversing the input or rotating it n times (Davida & Rodriques [6]). And finally, there are many files from which to choose as input. Following a minimum amount of analysis, it is apparent that LFSRs can be used in three basic ways for compression. To maintain tractability, we decided not to re-arrange the input prior to processing, though we experimented with this slightly (see Section 7), and we chose standard input sets, so-called compression corpora. We believe that this approach enables us to use our results to make conclusions about the use of LFSRs, in general, for compression.

To calibrate our experiments we ran the same input through current, standard compression algorithms, such as Huffman coding and LZW.

Massey describes one way of using LFSRs for compression in the passage cited at the beginning of this paper: the input is partitioned into 127 bit blocks, then each block is run through the Berlekamp-Massey algorithm to find a minimal-length LFSR that generates that block; the algorithm outputs the length of the LFSR, always using exactly 7 bits, followed by the initial setting for the LFSR and the settings for the tap sequence. We refer below to this way of using LFSRs for compression as "fixed" mode. We also consider what we call "variable" and "adaptive" modes.

In this Section we describe the three modes we used and show why we believe these represent the extent of using LFSRs for compression. Accordingly, we first consider the possible ways of using LFSRs for compression, then we prune that set to arrive at the three modes we use.

4.1 Approaches and Parameters

There are two basic approaches and two basic parameters.

The two approaches are adaptive and non-adaptive. Generally speaking, an adaptive approach changes parameter values during the course of a run based on the current data. A non-adaptive approach makes no such changes at run time.[1] The non-adaptive approach we present below devolves into several sub-approaches; the adaptive approach does not devolve.

The two parameters are input block size and LFSR size. The input block size is the length of the bit sequence for which we will ask the Berlekamp-Massey algorithm to find a corresponding LFSR. LFSR size is the length of the LFSR generated by the Berlekamp-Massey algorithm. We can control LFSR size only by observing the length of the LFSR as we feed the algorithm bits, then backtracking one bit when the LFSR exceeds the length we desire.

4.2 The Non-Adaptive Approach

In the non-adaptive approach we do not change parameters at run time. The length of the input bit sequence and the length of the LFSR can be either fixed or variable. This results in four options:

1. use a fixed size for both the input block and the LFSR;
2. use a fixed-sized input block and a variably-sized LFSR;
3. use a variably-sized block and a fixed-sized LFSR; or
4. use a variable size for both the input block and the LFSR.

---

1. "Variable" mode, as we describe it below, could be considered adaptive since the size that the approach determines for each input block is dependent on the nature of the input stream. However, parameter values do not change in variable mode as progression proceeds. Since the mode we describe as "adaptive" changes parameter values as the progression proceeds. we decided to use the words "variable" and "adaptive" to describe these two approaches, rather than less descriptive terms, such as "adaptive0" and "adaptive1."

These four options are shown in Table 1.

Table 1 Non-Adaptive Options

option	block-size	lfsr-size	method	comments
1	fixed	fixed	The lfsr-size must be set to the worst case, which is the length of the block size.	Unreasonable, since no compression is possible.
2	fixed	variable	Process each input block and generate the corresponding lfsr.	Trivially parallelizable. (This is Massey's suggested approach.)
3	variable	fixed	Process input until the lfsr that is built reaches the fixed size.	Is this efficiently parallelizable?
4	variable	variable	Choose the lfsr based on compression ratio.	

The first option is unreasonable because we would have to set the size of the LFSR to the worst case, which is the same length as the block size. This would result in expansion by a factor of 2, since we would require the length of the input bit sequence to describe the initial settings of the LFSR and another length of the input bit sequence to describe the taps. We can therefore dismiss this option.

The second option involves breaking the input into fixed-size blocks (padding the last block, as needed), then finding LFSRs for each block. Since the LFSRs will never be longer than the blocks, we can use a fixed number of bits to describe the length of the LFSR. This is a conceptually simple option. It is also trivially parallelizable. This is the approach Massey describes in the quote at the beginning of this paper.

The third option involves monitoring the length of the LFSR as it is being created. When the LFSR reaches a predetermined length (or we have no more input to consume, whichever comes first), we output the LFSR, then start building a new one with the next unused bit in the input bit sequence. This option is not as conceptually simple as the second option, and involves more overhead. It is not easily parallelizable because it is not possible to determine beforehand the length of the block that a given LFSR will represent. In addition, we see no reason to suppose that this would give better compression than the second option. For these reasons we anticipate the third option being uniformly inferior to the second, and have thus not pursued it.

The fourth option involves generating the LFSR based on the compression ratio. As we feed bits to the LFSR-generator, we monitor the compression ratio for each prefix of the entire input block, remembering the prefix that provided the best ratio. When the entire input block has been fed in, we choose the prefix that provides the best compression ratio, and we generate the corresponding LFSR. The beginning of the next input block starts one bit past the last bit generated by the previous LFSR. This option may require that we process the same bit many times. It is more complicated than the other options; it involves more overhead; and it is not parallelizable. However, it may provide sufficiently better compression to compensate for the additional overhead.

4.3 The Adaptive Approach

The adaptive approach is based on the assumption that the overall compression ratio will increase if the size of the input blocks steadily increases if the compression ratio is getting better, and vice versa if the compression ratio is getting worse.

We have constrained this approach to use block sizes that are one less than an integral power of 2. Therefore for this approach, we need three constants: the initial block size, and the minimum and the maximum block sizes. Given our constraint, it is simplest just to specify the representative power of 2 for each constant, thus for example, setting initial block size to 7 implies that the initial block size will be $2^7-1 = 127$.

Adaptive mode works as follows: assume that the size of the current block is s_i, for $i > 0$, and that the compression ratios for the previous block and the current block, are r_{i-1} and r_i, respectively, where compression ratio equals compressed output size divided by uncompressed input size. If $r_{i-1} \geq r_i$, then the compression is getting better (or at least getting no worse), so we set s_{i+1} to the next larger block size, namely $s_{i+1} = \min(2*(s_i+1)-1, 2^{\text{maximum\_block\_size}}-1)$; otherwise we set s_{i+1} to the next smaller block size, namely $s_{i+1} = \max(((s_i+1)/2)-1, 2^{\text{minimum\_block\_size}}-1)$.

The appeal of this approach is its promise of an advance over what Bell, Cleary, and Witten refer to as "ad hoc" compression methods: the adaptive approach applies to all data types, and thus research can focus on performance, instead of a continual search for new algorithms for new data types (Bell et al. [7]).

4.4 The Three Modes

The modes that we used in our experiments are Options 2, 4, and adaptive. Each has strengths

and weaknesses, as summarized in Table 2.

Table 2 Modes

mode	block-size	lfsr-size	method	strengths	weaknesses
Fixed (Option 2)	fixed	variable	Generate the LFSR when the input block has been consumed. (Massey's suggested approach.)	Simple, fast, and trivially parallelizable.	What criteria do we use in choosing the block size?
Variable (Option 4)	variable	variable	Generate the LFSR for the prefix of the block that gives the best compression ratio.	May provide the best compression.	Slow: each bit may need to be processed many times. What criteria do we use to determine the upper bound on the input block size? Not parallelizable.
Adaptive	adaptive	variable	Choose the size of the next input block based on the ratio of the compression ratio of the current block and the previous block.	Can adapt to the data.	What criteria do we use to determine the three constants? Not parallelizable.

5 Experiments

This section describes (a) the two parameter settings we used, (b) "bypass expansion," and (c) the nine trials that we ran. It also describes the standard control algorithms to provide relative performance and the set of input files we used.

5.1 Parameter Settings

We used two parameter settings, named 7 and 10. The first is the input block size that Massey suggests, namely $127 = 2^7-1$ bits. The second represents a larger block size, $1023 = 2^{10}-1$. The values for each parameter setting are shown in Table 3. (We did not pursue block sizes smaller than 127 because experiments suggested that compression performance decreased with smaller

block sizes.)

Table 3 Parameter Settings.

internal named parameter	parameter setting name		applicable mode
	7	10	
	block size values		
fixed_block_size	2^7-1 = 127	2^{10}-1 = 1023	fixed
variable_block_size			variable
initial_block_size			adaptive
min_block_size	2^4-1 = 15	2^7-1 = 127	
max_block_size	2^{10}-1 = 1023	2^{13}-1 = 8191	

5.2 Bypass Expansion

We experimented with bypassing expansion. If the size of the compressed output is greater than the input, then we will get better "compression" if we pass the raw input on as output. This capability requires an extra bit to indicate whether what follows is raw input or compressed data.

5.3 Trials

We ran each of the two parameter settings on each of the three LFSR modes. We also ran parameter setting 7 with bypass expansion on each of the three LFSR modes, using parameter setting 7. This makes for nine trials in all, as shown in Table 4.

Table 4 Trial Configurations.

trial number	mode	parameter setting	bypass expansion?
1	fixed	7	no
2	variable		
3	adaptive		
4	fixed		yes
5	variable		
6	adaptive		
7	fixed	10	no
8	variable		
9	adaptive		

5.4 Control Algorithms

To calibrate our experiments, we ran all of the input we used through current standard compression algorithms. We used the implementations provided in Nelson & Gailly's reference

on compression algorithms for Huffman, adaptive Huffman, four variations of arithmetic coding, two variations of LZW, and LZSS (Nelson & Gailly [8]). (The results of the Nelson & Gailly codes are shown in Appendix A.) We also created our own implementation of a simple, run-length encoding scheme, MNP5 (Salomon [9]).

5.5 Input Files

We chose standard input sets, so-called compression corpora, as input. We used the Canterbury Corpus, the Canterbury Corpus large set, and the older Calgary Corpus, all available on-line ([10]). The advantage of using these files is that they represent a standard.

We also used three graphics files named comet, grid, and wallball, each approximately 5 MB long, that are of particular interest to projects at our laboratories.

Information about the files in the compression sets we used is shown in Table 5. The numbers in the first column of the table correspond to the numbers used in the plots shown in Section 6; in order to open a gap between corpora in the plots, there are no items numbered 12, 16, or 35 in Table 5.

Table 5 Compression Corpora.

item number	Corpus	item name	size (bytes)	description
1	Canterbury	alice29.txt	152089	text (English text)
2		asyoulik.txt	125179	play (Shakespeare)
3		cp.html	24603	HTML
4		fields.c	11150	Csrc (C source)
5		grammar.lsp	3721	list (LISP source)
6		kennedy.xls	1029744	Excl (Excel Spreadsheet)
7		lcet10.txt	426754	tech (Technical writing)
8		plrabn12.txt	481861	poem (Poetry)
9		ptt5	513216	fax (CCITT test set)
10		sum	38240	SPRC (SPARC Executable)
11		xargs.1	4227	man (gnu manual page)
13	Canterbury (large set)	E.coli	4638690	Complete genome of the E. Coli bacterium
14		bible.txt	4047392	The King James Version of the Bible
15		world192.txt	2473400	The CIA world fact book

Table 5 Compression Corpora.

item number	Corpus	item name	size (bytes)	description
17	Calgary	bib	111261	Bibliography (refer format)
18		book1	768771	Fiction book
19		book2	610856	Non-fiction book (troff format)
20		geo	102400	Geophysical data
21		news	377109	USENET batch file
22		obj1	21504	Object code for VAX
23		obj2	246814	Object code for Apple MAC
24		paper1	53161	Technical paper
25		paper2	82199	Technical paper
26		paper3	46526	Technical paper
27		paper4	13286	Technical paper
28		paper5	11954	Technical paper
29		paper6	38105	Technical paper
30		pic	513216	Black and white fax picture
31		progc	39611	Source code in "C"
32		progl	71646	Source code in LISP
33		progp	49379	Source code in PASCAL
34		trans	93695	Transcript of terminal session
36	xwd graphics files	comet	5050491	an object striking a plane
37		grid		a background grid
38		wallball		a ball splashing into a soft wall

5.6 Calculating Output Size

In this section we describe how we calculated the size of each output block.

We require one bit to flag a block that is bypassed due to bypass expansion; when this option is turned off, this extra bit is not needed and is not included in the output. Since we do not truncate or pad input files, we flag the last block in every file and must include its size. The particulars are

shown in Table 6.

Table 6 Flag and Field Lengths.

flag/field name	mode		
	fixed	variable	adaptive
bypass_expansion_flag	1[a]		
last_block	1		2[b]
input_length	0; for last block, B[c]	B	0; for last block, B
lfsr_length	B		
lfsr	lfsr_length		
taps			

a. This one bit is needed only if bypass expansion is turned on.

b. This field combines two fields: a last_block flag and a size_of_next_block field. These 2 bits indicate whether the next block is (a) the last block in the file, in which case the length will be specified via the input_length field, or else the next block is (b) bigger than the previous block, (c) smaller than the previous block, or (d) the same size as the previous block. In the latter three cases, there is no need for the input_length field.

c. The notation "B" represents the number of bits required to represent the maximum, current input block length. For example, if the input block length is 127, then B = 7. B must represent the maximum length so that the decoder knows the number of bits to interpret for this field. B must also represent the current length because under adaptive mode the length changes.

As an example, the field sizes and resulting compression ratio for fixed, variable, and adaptive modes for a sample input block of size 127, with bypass expansion turned off, are shown in Table 7. The Table shows that for these parameters and this input block, the best compression is provided by variable mode; fixed mode provides the slightest edge in better performance over adaptive, but both expand the input block. However, note that variable mode only compressed the first 66 bits; the remaining 61 bits of the block would be part of the next block that variable mode would consider.

Table 7 Example Output Block.[a]

name	mode		
	fixed	variable	adaptive
bypass_expansion_flag	0		
last_block	1		2
input length	0	B(127) = 7	0
lfsr_length	B(127) = 7		
lfsr	60	16	60
taps	60	16	60
TOTAL	128	47	129
Compression ratio	128 / 127 = 1.00787	47 / 66 = 0.712121	129 / 127 = 1.01575

a. The values in this Table are for the first block for Trial 1 (see Table 4), which uses parameter setting 7 (see Table 3), on the first item in our input set, file "alice29.txt" (see Table 5).

6 Results

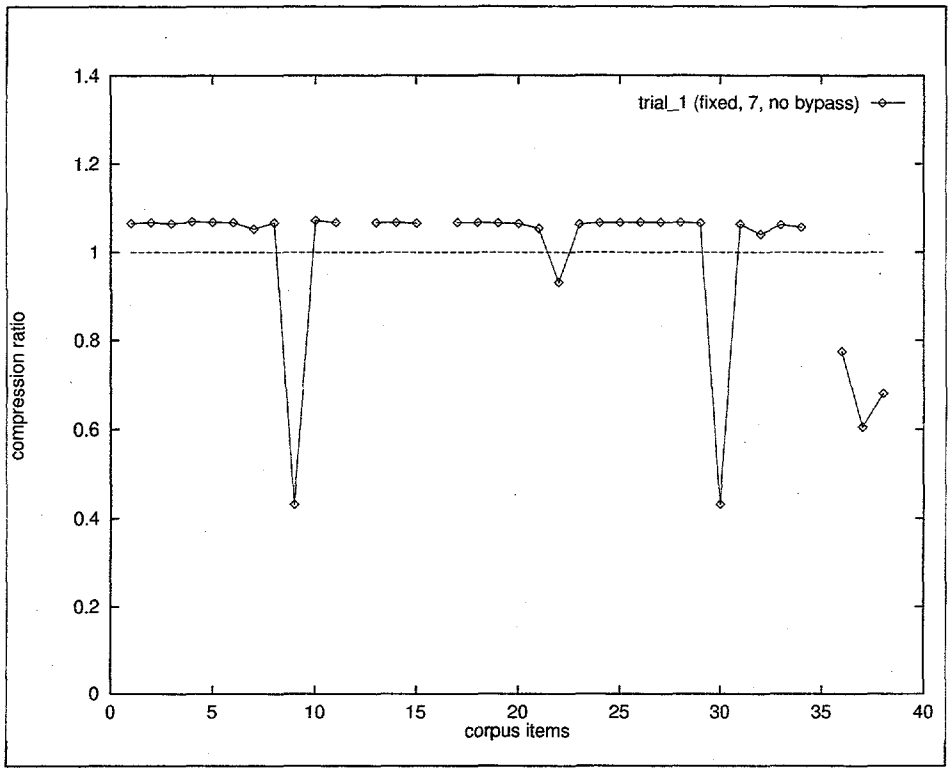
The results of the trials are shown in this section in a series of plots that indicate LFSRs are not in the same league as current standard algorithms. In fact, they are barely in the same league as a simple, run-length encoding algorithm.

The first plot shows the results for Trial 1—fixed mode, using parameter setting 7. For ease of reference in this plot and subsequent ones we have drawn a dashed line at y=1: points below this line represent compression; points above this line represent expansion. We have also connected the values with lines; these lines do not imply that the values represent points of a continuous function; rather, the lines will help identify different sets of values when they are graphed on the same plot, as we will show in subsequent plots. The input files are numbered according to Table 5, and we have included a gap between corpora, all for ease of reference.

This first plot shows that Trial 1 results in expansion for all of the files in our input set except for files numbered 9, 30, 36, 37, and 38—ignoring file 22 since it is so close to 1. These files are ptt5 (fax (CCITT test set), pic (another fax), and all three of our graphics files, respectively. The same general pattern in the results for Trial 1 is apparent in the results for the other trials. We conclude that LFSRs provide better compression for graphics files than they do on text files, but even at its best, LFSRs produce only moderate performance. For example, the compressed output for file 9,

the best for Trial 1, is no better than 0.4 the size of the input.

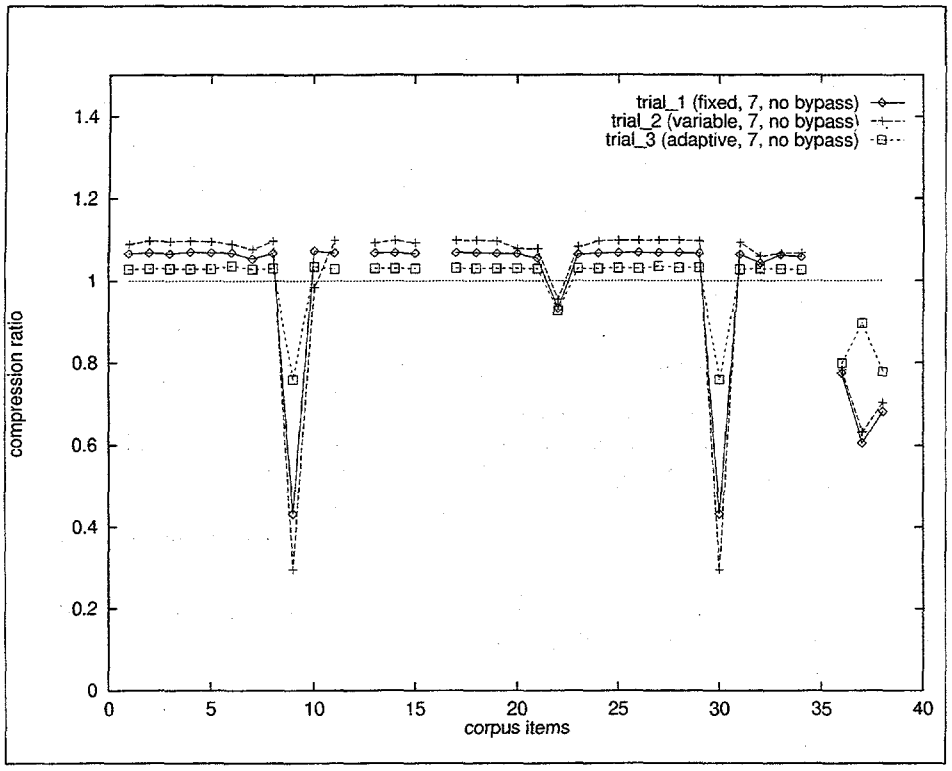
Figure 1. Trial 1: fixed mode, parameter setting 7, no bypass expansion.



The second plot shows the results for Trials 1, 2, and 3—for the trials using parameter setting 7 without bypass expansion. When the files compress, variable mode provides the best performance, followed by fixed mode, followed by adaptive mode. However, when the files do not compress, the roles reverse: adaptive provides the best performance, followed by fixed, followed by variable. Fixed mode could be said to provide the best all-around performance.

Recall that fixed mode has lower overhead than either of the other two modes.

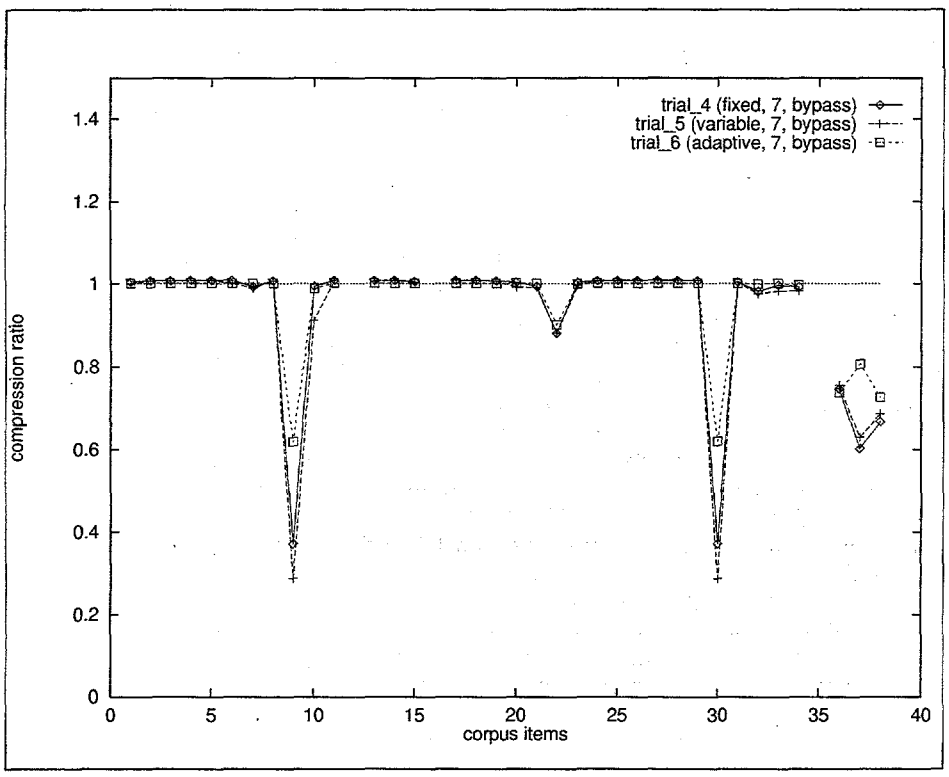
Figure 2. Trials 1, 2, and 3: parameter setting 7, no bypass expansion.



The third plot shows the results for Trials 4, 5, and 6—for the trials using parameter setting 7 and bypass expansion. We note that the performance is better here than it was for Trials 1-3 but the

relationship of performance is the same as it was for Trials 1-3.

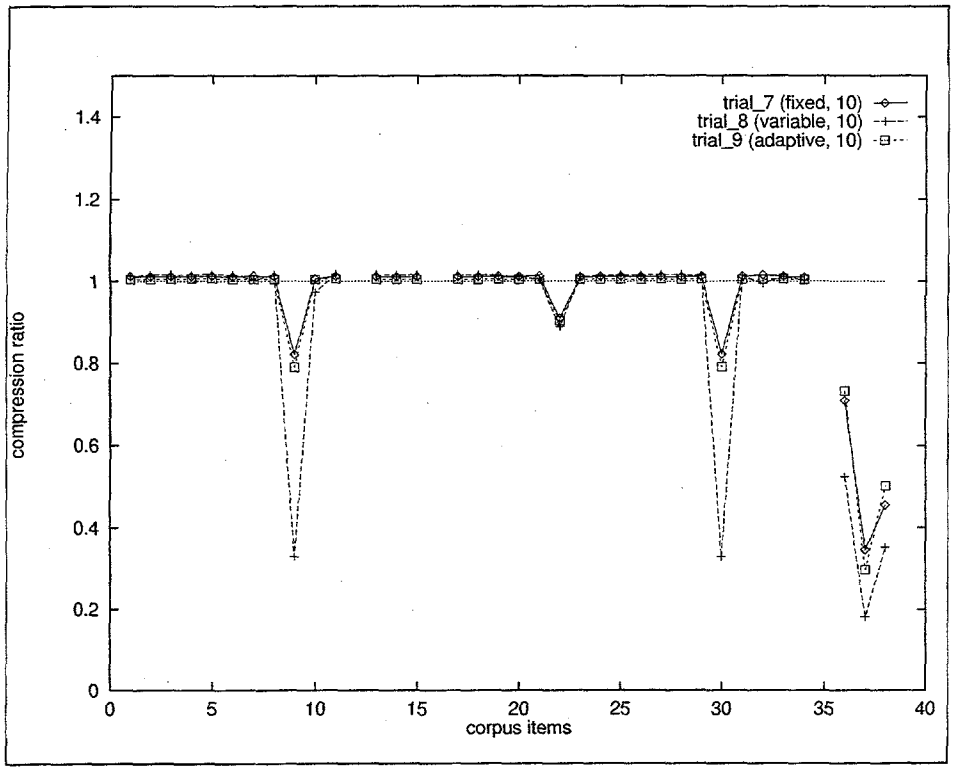
Figure 3. Trials 4, 5, and 6: parameter setting 7, bypass expansion.



The fourth plot shows the results for Trials 7, 8, and 9—for the trials using parameter setting 10 without bypass expansion. Performance appears to be even better than with the previous six trials. However, unlike the other trials in which fixed mode provides the best compression, variable mode provides the best compression here. The difference is due primarily to the larger

block size, which decreases the relative size of the overhead required for variable mode.

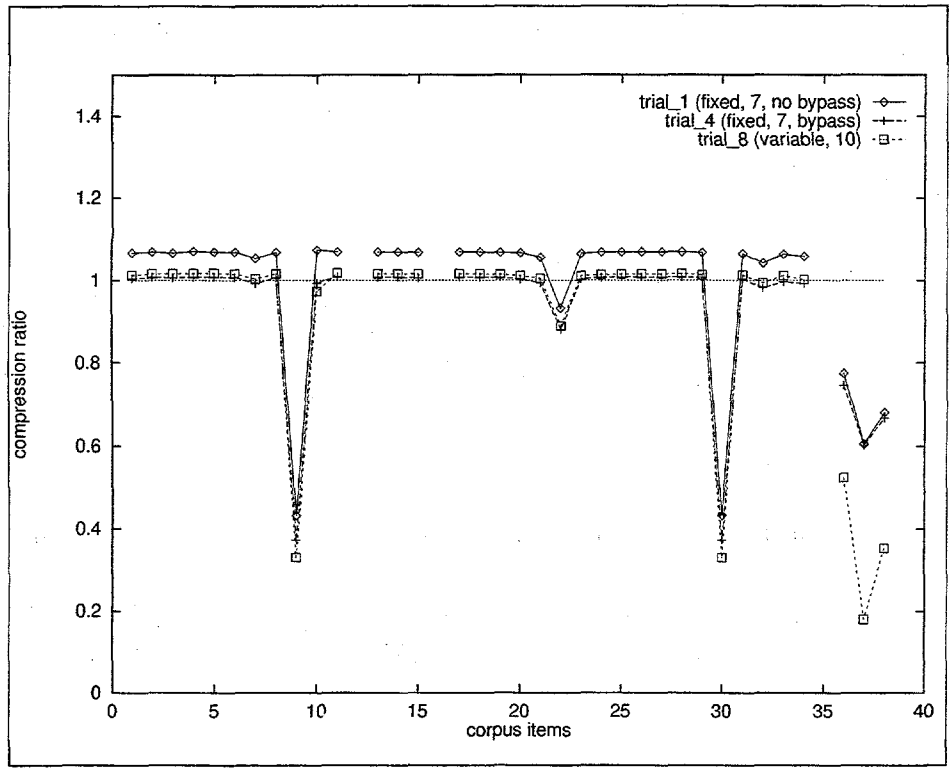
Figure 4. Trials 7, 8, and 9: parameter setting 10.



The fifth plot shows the best trials of the previous three plots—Trials 1, 4, and 8. The plot shows that variable mode using parameter setting 10 provides the best performance, although not by a significant amount in most cases. Bypassing expansion in fixed mode provides almost the same performance that increasing the block size and changing to variable mode provides, except for

input items 36-38.

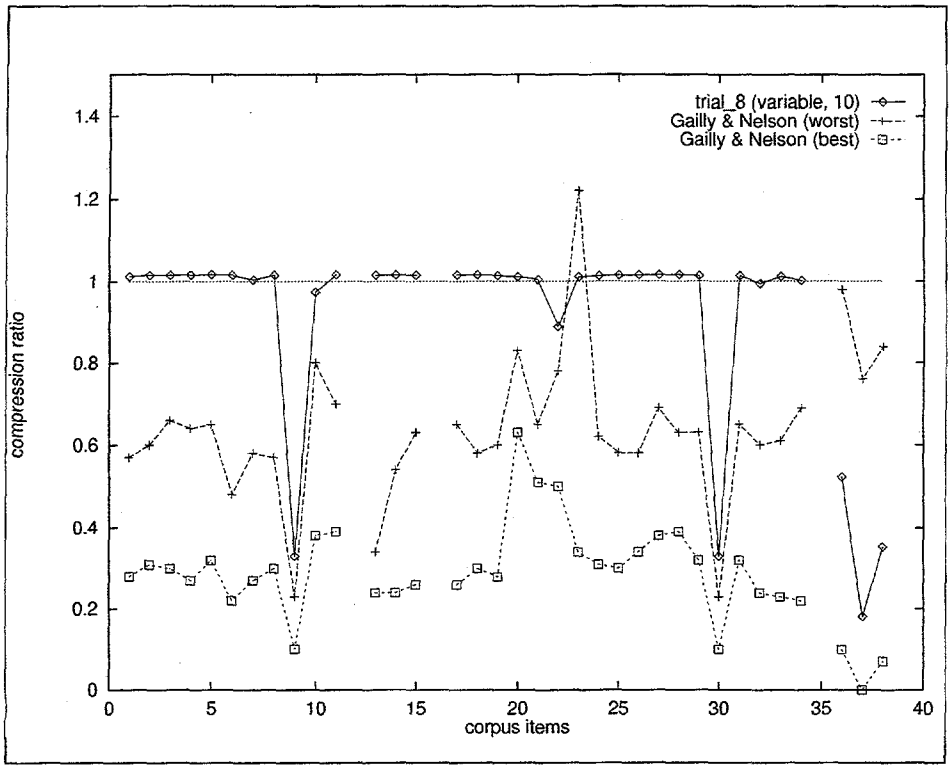
Figure 5. Best of Trials <1,2,3>, <4,5,6>, and <7,8,9>: Trials 1, 4, 8.



The sixth plot compares the results of Trial 8, the best of the nine trials, and the worst and best of the algorithms provided by Nelson & Gailly. Trial 8 is never better than the best of the Nelson & Gailly algorithms. Only on input items 23 and 36-38 is Trial 8 better than the worst of the Nelson & Gailly algorithms. This plot shows that LFSRs cannot provide the same performance

as current, standard algorithms, at least not on the current, standard input files.

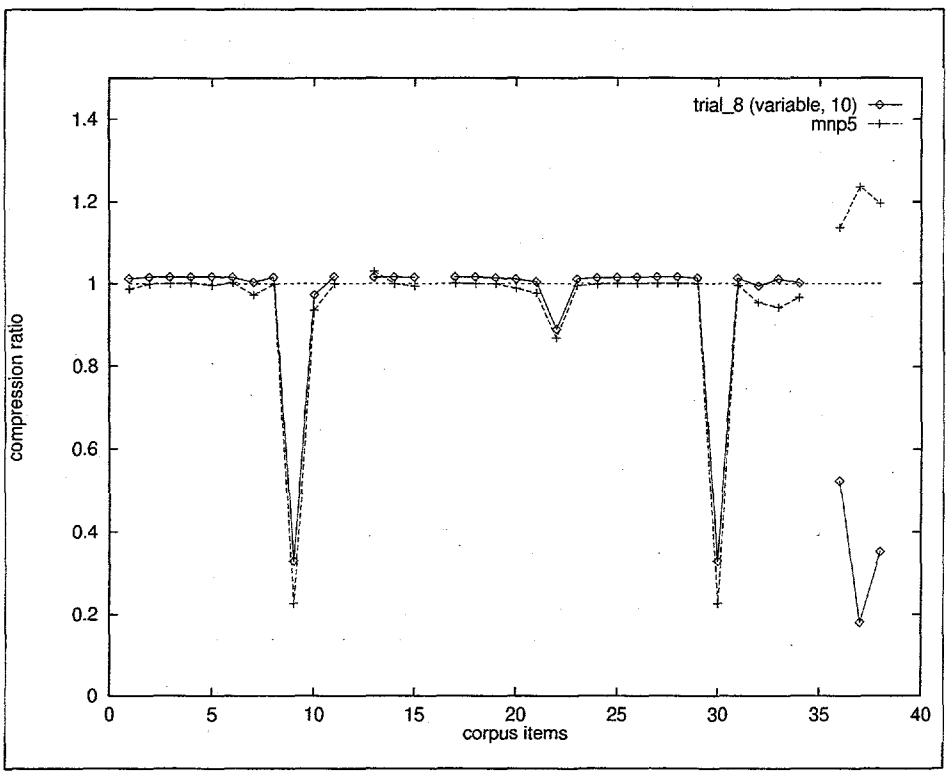
Figure 6. Best of LFSR (Trial 8: variable, 10); Worst and Best of Gailly & Nelson.



The seventh plot compares the results of Trial 8 and MNP5. The plot shows that the two algorithms are comparable, except for the input files 36-38, for which LFSR is qualitatively

better.

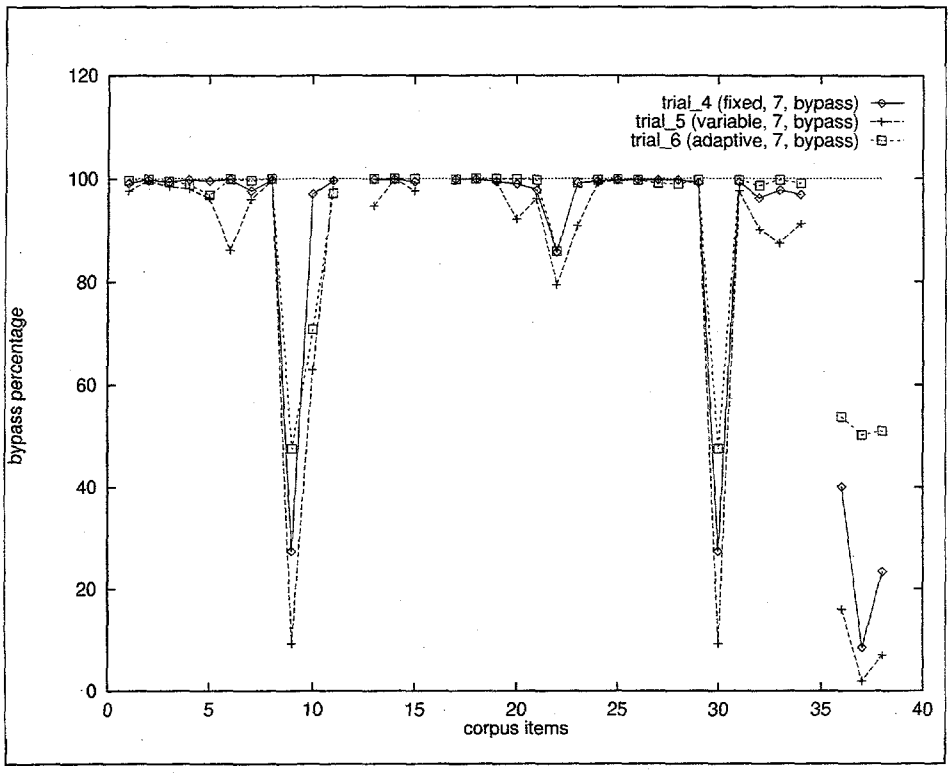
Figure 7. Trial 8, and MNP5.



The eighth and final plot shows the percentage of blocks that were bypassed in Trials 4, 5, and 6. Bypassed blocks are those for which the compressed output is no smaller than the input. For these blocks the raw input is presented as the "compressed" output. Note that for most of the files, the bypass percentage is almost 100%, indicating that few of the input blocks could be compressed and providing an explanation for the poor compression in general available via

LFSRs.

Figure 8. Bypass Percentage for Trials 4, 5, and 6.



7 Related Work

The only work of which we are aware that has pursued this topic is Davida & Rodriques' [6]. Their results show that LFSRs provide compression that is "comparable to the other compression algorithms such as Adaptive Huffman, Lempel-Ziv-Welch, etc." Davida & Rodriques use many heuristics, none of which we use. However, for purposes of comparison we did experiment briefly with just two of the techniques that they used—reverse and rotate. For reverse, we found the shortest LFSR for both the original input string and for its reverse, then we used the best result. This approach requires an extra bit in the output to indicate whether the LFSR generates the original input string or its reverse. For rotate, we found the LFSR for the original input string and for each rotated string. The number of rotated strings is one less than the length of the string, so this approach requires $\log_2$(string_length) extra bits in the output. We ran reverse and rotate on the Canterbury Corpus, using Trial 1 settings; the results indicate that on average rotate boosts

compression by less than 1%, and reverse degrades compression (for 126 times the effort), as shown in Table 8.

Table 8 Percent Improvement in Compression Ratios for Reverse and Rotate on the Canterbury Corpus (Trial 1 settings).

approach	Canterbury Corpus (percent change in compression ratio)		
	best	worst	average
reverse	-8%[a]	0.6%	-0.4%
rotate	-12%	4%	0.03%

a. That is, when we run the reverse option on the Canterbury Corpus using Trial 1 settings, the best improvement we get for compression ratio is 8% better than without that option.

The previous Table shows aggregate performance. This next Table looks at what we believe is a representative sample, the first five entries in the Canterbury Corpus. Only the first entry shows improvement, and then only when we use reverse, as shown in Table 9.

Table 9 Percent Improvement in Compression Ratios for Reverse and Rotate on initial entries in the Canterbury Corpus (Trial 1 settings).

approach	Canterbury Corpus item name (percent change in compression ratio)				
	alice29.txt	asyoulik.txt	cp.html	fields.c	grammar.lsp
reverse	-0.03%[a]	0.26%	0.31%	0.16%	0.53%
rotate	0.55%	0.83%	0.88%	0.67%	0.87%

a. That is, when we run the reverse option for alice29.txt under fixed mode using parameter setting 7, we get a compression ratio that is 0.03% better than without that option. Note that this is the only entry in the table that shows improvement; all the rest show a degradation of performance.

Acknowledgments

We would like to thank Anne Van Arsdall for her review of this paper.

References

[1] J. L. Massey, "Shift-Register Synthesis and BCH Decoding." *IEEE Tran. Information Theory* IT-15, 122 (1969).

[2] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Florida, 1997.

[3] F. G. Gustavson, "Analysis of the Berlekamp-Massey Linear Feedback Shift-Register Synthesis Algorithm," *IBM Journal of Research and Development*, 20:204-12, May 1976.

[4] P. L. Campbell, "An Implementation of the Berlekamp-Massey Algorithm in C," SAND 99-2033. August 1999. Sandia National Laboratories, Albuquerque, NM.

[5] B. Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons, New York, 1996.

[6] G.I. Davida, L. Rodriques, "Data Compression Using Linear Feedback Shift Registers," Data Compression Conference, Snowbird, Utah. March 29-31, 1994.

[7] T. C. Bell, J. G. Cleary, I. H. Witten, *Text Compression*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[8] M. Nelson, J.-L. Gailly, *The Data Compression Book*, 2nd Edition. M&T Books, New York, New York. 1996.

[9] D. Salomon, *Data Compression: The Complete Reference*. Springer-Verlag New York, Inc., New York. 1998.

[10] The Canterbury Corpus and large set: http://corpus.canterbury.ac.nz/fileset.html.

Appendix A Nelson & Gailly Results

This Appendix shows the compression ratio (output size / input size, expressed as a percentage) for each of the implementations presented in Nelson & Gailly on each of the input files that we used. For details on (and source for) each of the algorithms, see Nelson & Gailly's text. [8]

Table 10 Compression Ratio for Nelson & Gailly Codes on the Canterbury Corpus.

Corpus item	Nelson & Gailly Code								
	huff	ahuff	arith	arith1	arith1e	arithn	lzw12	lzw15v	lzss
alice29.txt	0.57	0.57	0.57	0.46	0.43	0.28	0.47	0.41	0.49
asyoulik.txt	0.60	0.60	0.60	0.47	0.43	0.31	0.50	0.43	0.53
cp.html	0.66	0.66	0.65	0.57	0.49	0.30	0.49	0.46	0.45
fields.c	0.64	0.64	0.63	0.58	0.44	0.27	0.47	0.44	0.35
grammar.lsp	0.61	0.60	0.60	0.65	0.47	0.32	0.56	0.48	0.42
kennedy.xls	0.48	0.43	0.46	0.34	0.32	0.22	0.40	0.28	0.33
lcet10.txt	0.58	0.58	0.58	0.45	0.44	0.27	0.52	0.41	0.47
plrabn12.txt	0.57	0.57	0.56	0.43	0.42	0.30	0.48	0.44	0.56
ptt5	0.23	0.20	0.21	0.11	0.10	0.10	0.13	0.12	0.21
sum	0.69	0.67	0.68	0.56	0.50	0.38	0.80	0.52	0.47
xargs.1	0.64	0.63	0.63	0.70	0.53	0.39	0.63	0.55	0.50

Table 11 Compression Ratio for Nelson & Gailly Codes on the Canterbury Corpus (large set).

Corpus item	Nelson & Gailly Code								
	huff	ahuff	arith	arith1	arith1e	arithn	lzw12	lzw15v	lzss
E.coli	0.28	0.27	0.25	0.25	0.24	0.24	0.27	0.27	0.34
bible.txt	0.54	0.54	0.54	0.41	0.40	0.24	0.45	0.36	0.41
world192.txt	0.63	0.62	0.62	0.46	0.45	0.26	0.63	0.44	0.54

Table 12 Compression Ratio for Nelson & Gailly Codes on the Calgary Corpus.

Corpus item	Nelson & Gailly Code								
	huff	ahuff	arith	arith1	arith1e	arithn	lzw12	lzw15v	lzss
bib	0.65	0.65	0.65	0.48	0.43	0.26	0.48	0.41	0.48
book1	0.58	0.56	0.56	0.46	0.45	0.30	0.50	0.45	0.56
book2	0.60	0.59	0.60	0.46	0.47	0.28	0.56	0.43	0.48
geo	0.71	0.71	0.71	0.63	0.65	0.63	0.76	0.76	0.83
news	0.65	0.65	0.64	0.54	0.52	0.63	0.61	0.51	0.53

Table 12 Compression Ratio for Nelson & Gailly Codes on the Calgary Corpus.

Corpus item	Nelson & Gailly Code								
	huff	**ahuff**	**arith**	**arith1**	**arith1e**	**arithn**	**lzw12**	**lzw15v**	**lzss**
obj1	0.76	0.76	0.76	0.65	0.63	0.50	0.78	0.65	0.57
obj2	0.79	0.77	0.78	0.55	0.53	0.34	1.22	0.53	0.42
paper1	0.62	0.62	0.62	0.54	0.48	0.31	0.58	0.47	0.47
paper2	0.58	0.57	0.57	0.50	0.45	0.30	0.50	0.44	0.49
paper3	0.58	0.58	0.58	0.53	0.47	0.34	0.51	0.47	0.51
paper4	0.69	0.59	0.59	0.60	0.50	0.38	0.56	0.52	0.52
paper5	0.63	0.63	0.62	0.63	0.52	0.39	0.58	0.55	0.52
paper6	0.63	0.63	0.62	0.56	0.49	0.32	0.61	0.49	0.47
pic	0.23	0.20	0.21	0.11	0.10	0.10	0.13	0.22	0.21
progc	0.65	0.65	0.65	0.57	0.49	0.32	0.61	0.48	0.45
progl	0.60	0.59	0.59	0.47	0.41	0.24	0.48	0.37	0.32
progp	0.61	0.61	0.61	0.49	0.48	0.23	0.47	0.38	0.32
trans	0.69	0.69	0.69	0.49	0.44	0.22	0.53	0.40	0.36

Table 13 Compression Ratio for Nelson & Gailly Codes on the XWD Files.

Corpus item	Nelson & Gailly Code								
	huff	**ahuff**	**arith**	**arith1**	**arith1e**	**arithn**	**lzw12**	**lzw15v**	**lzss**
comet	0.57	0.49	0.57	0.27	0.27	0.10	0.98	0.18	0.29
grid	0.23	0.17	0.24	0.09	0.08	~0.0	0.76	0.01	0.12
wallball	0.38	0.35	0.38	0.18	0.18	0.07	0.84	0.12	0.25

Distribution List

2	MS	0449	P. L. Campbell	Org.	6236
1		0449	R. L. Hutchinson		6236
1		0451	R. E. Trellue		6202
1		0612	Review & Approval Desk For DOE/OSTI		4912
1		0741	S. G. Varnado		6200
1		0806	L. B. Dean		4616
1		0806	L. G. Pierson		4616
1		0806	T. D. Tarman		4616
1		0806	M. O. Vahle		4616
1		0806	E. L. Witzke		4616
1		0874	P. J. Robertson		1716
1		0874	K. L. Gass		6421
2		0899	Technical Library		4916
1		8903	P. W. Dean		9011
1		9018	Central Technical Files		8940-2