A COMPARISON OF

TEXT COMPRESSION METHODS.

V.J. PAUL.
DEPARTMENT OF COMPUTER SCIENCE.
UNIVERSITY OF CANTERBURY.
OCTOBER, 1976.

CONTENTS

1 INTRODUCTION.

Text compression is the activity of taking a file of text and subjecting it to a compaction process to reduce its storage requirements. Text compression is of interest because of the many applications involving the storing of large amounts of text. No matter how much secondary storage is available it is still expensive and invariably there are other requirements for its use.

Apart from the obvious advantage of reducing the amount of storage required, there are other possible benefits, for example:

(i) A degree of security for the file, as output from a compression routine is usually in an unreadable form.

(ii) Savings in data transmission costs, since the cost for transmission is usually proportional to the quantity of data transmitted.

(iii) A reduction in the time to run programs in many cases, especially if little computation is involved compared with the amount of input and output.

This investigation of text compression has involved:

(i) Considering the situations for which text compression might be useful, and the criteria by which alternative text compression methods should be evaluated.

(ii) A discussion of several different methods.

(iii) Implementing and testing some of these methods and examining the results in the light of results obtained by others and where possible against theoretically expected results or limits.

2 SITUATIONS.

General areas where text compression techniques could be applied include.

(i) Error message routines for compilers, interpreters and interactive systems.

(ii) Archiving of source programs or other information.

(iii) Data base management systems or other large information storage and retrieval systems.

Text compression techniques are most suitable for large files containing homogeneous data. The advantages are greater for random access files because the cost of expanding a record is likely to be small compared with the cost of a random access to it. For example many on line inquiry terminal systems fit these characteristics.

Where the application for which text compression is being considered is I/O bound, text compression could be applied with no increase in elapsed time due to extra CPU time to expand the records.

This is just one of the criteria that should be considered.

The situations investigated here are:

(i) The storage and retrieval of Algol source programs.

(ii) The storage and retrieval of the error messages for the PDP 11 Basic interpreter.

3 CRITERIA

The criteria by which a particular compression method should be evaluated for a particular application are, in decreasing order of importance:-

1. Compression Ratio.
2. Characteristics of data.
3. Decompression time.
4. Size of decompression alg rithm and tables.
5. Compression time and size of compression algorithm and tables.
6. Complexity of algorithms.

In more detail the meanings of and reasons for choosing these criteria are set out below.

1. THE COMPRESSION RATIO

This can be expressed as a fraction or a percentage, the figure used in this report is

$$CR = \frac{100}{1} \quad X \quad \frac{\text{Output file (bytes)}}{\text{Input file (bytes)}}$$

In this calculation the space required for algorithms or tables should not be included.

This should only be considered after the size of the file to be compressed is known. However what should be included is any overhead associated with each record. For example variable length records usually require one extra word per record. Such overhead should be included in the size of both the input and output files.

2. CHARACTERISTICS OF DATA

Characteristics that make a file particularly suited to compression techniques include;

(i) Use of only a relatively small number of characters in the coding system. For example the Basic error messages use only 29 of the 96 ASCII printing characters.

(ii) Using most frequently a relatively small proportion of the character set. For example in the Algol source programs characters like "b" were used commonly but ones like "?" and "|" very rarely.

(iii) Patterns of characters occuring frequently in the text. For example reserved words in a computer program source file.

(iv) Fields of identical characters such as blanks or zeroes.

(v) The file contents do not change much over time: For example a list of addresses.

3. DECOMPRESSION TIME

Data will have to be processed by a decompression algorithm each time it is required for printing out and often for other use in memory. Therefore the decompression time is of ~~prime~~ importance, particularly if the program is running in a multi-programming enviroment.

4. DECCOMPRESSION ALGORITHM

The size of this and its tables is something that should be considered if there is doubt about whether it ia worthwhile implementing a compression/decompression scheme. The compression ratio should be re-evaluated taking into account the algorithm and tables as part of the output data.

5. COMPRESSION ALGORITHM

While most systems will be mainly retrieving data, it also needs to be stored, often this is done, ~~is~~ i.e. the more volatile the file, the more important it is that the Compression Algorithm be reasonably efficient in its use of memory space and CPU time.

6. COMPLEXITY.

If a compression method is being considered to save money, either directly or indirectly, then costs of development of the compression/decompression system should be considered. To some degree, the complexity of a method will be reflected in the costs of development, so relative simplicity or complexity of different methods should be taken into account.

4 METHODS.

The listed References include discussion of five substantially different text compression methods, and of two variations on one of the methods - pattern substitution.

In addition I have devised a method not mentioned in any of the references (Section 4.6)

4.1 Blank Suppression.

The most obvious and simplest scheme is to eliminate leading and trailing blanks in a record, and to leave the rest of the text uncompressed. This causes the problem common to all text compression schemes of producing variable length records that require extra overheads to manage. The system described by Fajman and Borgelt (2) used basically this strategy, but also compressed internal blanks from records.

To help overcome the problem of variable length records, they used the following method.

(i) Each line of text is divided into segments that can describe up to 15 blanks followed by up to 15 non-blank symbols.

For example.

The text bbbbb THIS IS AN EXAMPLE. bbbbb ANOTHER ONE

Would be described |5|15|THIS IS AN EXAM|0|4|PLE.|5|12|ANOTHER ONE|

(ii) The line number and a count of the total number of bytes in all the segments is placed in front of each line.

(iii) The lines are combined into pages of a length appropriate for the device being used. A count at the start of each page gives the total number of bytes in the page.

(ii) and (iii) make accessing a given line number simple. Savings of over 50% in disk space are reported for this method.

The elimination of leading and trailing blanks can also be incorporated effectively in other text compression methods.

4.2 Pattern Substitution.

Another common method is pattern substitution, in which strings of characters that occur frequently in the text are replaced by some shorter, coded representation.

PUFFT (the Purdue University Fast Fortran Translator) (6) uses this method for storing its error messages. In this application, the idea is taken to its limit in that all words are assigned a code.

Decisions to be made for this method are:

(i) Common phrases to extract from the source text. Factors to consider are the length of the phrase and the frequency of its occurence in the data. Usually the common phrases will be words or groups of words but that need not be so. Mayne and Jones (4) describe an experimental system that dynamically selects its own dictionary. Many of its entries are partial words, commonly being prefixes or suffixes.

(ii) Method of reducing an input string.

For example.

Input COMPRESSION.

Dictionary Holds

WORD	CODE
COMP	%1
COMPRE	%2
ESSION	%3

A simple left to right scan to replace the longest string contained in the dictionary by its code produces %2SSION, while the optinal compression is in fact %1R % 3. Wagner (9) gives an optimal compression algorithm using integer programming principles.

(iii) The code to replace the common characters.

If, as in the above example, special characters are used, we have to be sure the character will not occur in the text in another context. Assuming an eight bit character representation, the number of different patterns is limited to 256. If, instead of a special character, the codes are bit patterns not used for the character set then we are restricted even further to about 160 patterns assuming the EBCDIC character set. However it is now worthwhile including patterns of only two characters in the dictionary of common patterns.

Experiments were conducted with a pattern substitution algorithm that used a preselected dictionary, the largest-first logic for reducing input, and non- EBCDIC codes to substitute for patterns

Wagners algorithm was also tested; this uses a preselected dictionary, integer programming logic for compression, and special characters for patterns.

4.3. Huffman Codes

Morse code uses shorter codes for more common characters and longer ones for the less common. This idea can also be used in text compression to reduce the expected length (entropy) of a coded character from its usual fixed length of 7 or 8 bits.

In Morse, a longer pause is used to delimit characters but for variable length computer codes the way of distinguishing individual characters is to require the code to have the prefix property. That is the code for any character is not duplicated as the beginning of a longer code for some other character. For example If the code for "E" is 011 then no other character has a coded representation beginning 011.

The Huffman codes (5) satify this requirement and have been used satisfactorily as the basis of a text compression scheme (7). Huffman codes also have the property of being optimal - data encoded using these codes could not be expressed in fewer bits.

A compression and decompression routine was programmed to investigate Huffman codes for text compression, and a Huffman coding scheme was worked out for both of the data sets being considered. For the Algol source text, 64 codes were required, and these ranged from 4 to 18 bits. The entropy of the code was 4.7 bits compared with EBCDIC's 8. For the Basic error messages, 30 codes were required. They ranged from 3 to 10 bits in length with an entropy of 4.3 bits.

A possible disadvantage of this method is that if the contents of the file alter significantly we may no longer have an optimal code.

4.4. *The fixed point number method*

Another method based on the frequency of occurrence of characters in the text is the numerical method described by Hahn (3)

This method removes leading and trailing blanks from a record and encodes the remaining characters, in groups of length N, as fixed point numbers. Each of the symbols in a group is looked up in a dictionary holding B symbols. Suppose that P_i *is the position of the ith symbol of a group* ($1 \leq i \leq N$) *in the dictionary* ($1 \leq P_i \leq (B-1)$). *The* B^{th} *position is used for an 'escape' character to permit an extension of the dictionary. A group of symbols with positions* P_1 , P_2 , , P_N *is encoded as the unique fixed point number* $P_1 * B^{N-1} + P_2 * B^{N-2} + \quad + P_{N-1} * B + P_N$. *More that B-1 symbols can be used by the use of the escape character - usually coded as zero. The escape character signifies to the decompression routine that the symbol lies in the dictionary in the range B+1 to 2B-1. More than one escape character may be used to extend the dictionary even further. In general if P is the position of some symbol in the dictionary, the symbol is encoded as INTEGER (P/B) escape characters followed by MOD (P, B).*

Note. INTEGER (X) denotes the integer portion of the real number X. MOD (X,Y) denotes X Modulo Y.

For example if B=21, N=7 and the symbols to be encoded had positions 5,7,20,25,17,1,... then the first number produced by the compression process would be $5*21^6 + 7 * 21^5 + 20*21^4 + 0*21^3 + 4*21^2 + 17*21 + 1$

The characters in the dictionary are ordered so that the most frequently occuring in the input text is in location 1 and the least frequent in the last location. This helps reduce dictionary search times and minimizes the number of escape characters needed.

The value of B or any value of N we wish to consider can be found simply. It is the largest integer value B such that $B^N - 1 \leq L$ *where L is the largest integer that can be stored in one word on a particular computer. On the Burroughs B6700,* $L \doteq 5.49 \times 10^{11}$. *Table 1 lists the values of N for the B6700's 6 character word. This method is not as effective on the Burroughs which uses only 39 of 48 bits to represent an integer, in contrast to IBM equipment which uses all 32 bits. To illustrate this the values of B allowable if Burroughs used all 48 bits to represent integers are included in Table 1.*

TABLE 1: Optimum Values of B for Given N Values.

N	*B*	*B (if 48 bits were used to represent integers)*
7	47	109
8	29	61
9	20	38
10	14	26
11	11	20
12	9	16
13	7	12

The removal of leading and trailing blanks entails an overhead of 1 word per record and in this is stored the number of leading blanks and the number of significant text characters following them.

Hahn says that this method is best suited to read - only files. It can be seen that a change in a record could mean the re-organisation of the whole file for certain cases.

4.5 The Combining Characters Method

The last of the methods found in the literature is that described by Synderman and Hunt (8). Their scheme takes advantage of two of the characteristics of much textual data discussed in section 3. These are the use of only a few of the possible bit patterns to represent characters and the differing frequencies of occurrence of characters.

The EBCDIC code for characters, which is scattered from 40_{16} to $F9_{16}$ is replaced by a compacted code in the range 00_{16} to $3E_{16}$. The remaining code configurations are used to represent pairs of characters in the following manner:

(i) A certain group of characters, usually the most frequently occuring and/or the vowels, are designated 'master characters' and each assigned a base address.

(ii) Another larger group of characters is designated as 'combining characters'. It includes all the master characters.

(iii) When the compacted code is assigned, the master characters are assigned the numerically lowest codes and the rest of the combining characters the next lowest.

(iv) As input text is being processed all characters are translated to the compacted code and then each character is examined to see if it is a master. If it is and the next character is a combining one, then the code for the combining character is added to the base address assigned to the master character and this value is stored in one byte. A character not combined with another in this fashion is stored in a byte of its own.

(v) On output, if a byte has a value greater than the highest compacted code value (3E in above example) then it represents a pair of characters and the value can be used for a table look-up. If not, the value is translated back to EBCDIC.

The product of the number of master characters and the number of combining characters must be less than or equal to the number of unused code configurations. This still leaves scope for choosing the two numbers however. For example, in the Basic error messages only 29 characters are used, leaving 227 vacant positions. Possible master/ combining character arrangements are presented in table 2.

TABLE 2: Valid Master/Combining Character Arrangements For Synderman and Hunts Method Applied to the Basic Error Messages.

No. Master.	No. Combining.	Total ≤ 227
8	28	224
9	25	225
10	22	220
11	20	220
12	18	216
13	17	221
14	16	224

After trial and error testing, Synderman and Hunt decided to use the vowels and most frequently occuring characters as their master characters. In my program to test the method, the most frequently occuring characters were used regardless of whether they were vowels or not.

4.6. *An Alternative Method.*

An alternative text compression method, devised by the author, was drawn from a data communications technique discussed by Dr M.A. Maclean.

Here, a particular bit pattern may have more than one meaning, with the character it represents depending also an what 'mode' the decompression routine is in. One or more of the bit patterns has to mean 'change mode'. If there are more than two modes, it is also necessary to specify which one to change to.

This method was not considered for the Algol source text as there were 63 different characters and if the 8- bit EBCDIC code was replaced by a 5 - bit code (difficult to implement) two coding systems would be required. One change mode symbol common to both codes would be necessary. This implies we could reference only $2 * (2^5 - 1) = 62$ *characters. The situation is worse for a four bit code.*

However for the Basic error messages only 29 symbols are used and if a 4-bit code is used with 2 coding schemes then it is possible to reference $2 * (2^4 - 1) = 30$ *symbols. This is also easy to implement on the PDP-11 because it has a word length of 16 bits.*

The method is obviously very limited in the range of its applications, as it requires that the most convenient number of bits to represent the code is also a divisor of the word length to make implementation easy.

All the methods mentioned in this section have been implemented, and the findings from experiments are set out in the next section.

5. EXPERIMENTAL RESULTS

Table 3 gives an idea of the differences and similarities between the various algorithms written for the B6700

TABLE 3: Statistics For Algorithms

	1 Huffman Coding Method	2 Combining Characters Method	3 Fixed Point Number Method (Hahn)	4 Pattern Substitution Method	5 Wagners Algorithm
Total No. Cards.	169	130	214	160	127
Cards for compression	61	50	108	62	100
Cards for expansion	68	47	75	50	-
Core code estimate (Words)	2267	2132	2518	2207	7596
CPU compiletime (Secs)	2.57	2.35	2.76	2.34	7.22
Best Compression Ratio					
- Algol Source Data	54.6%	57.6%	39.2%	34.3%	84.9%
- Basic Error Messages	48.2%	53.9%	54.2%	46.3%	87.6%
- Other Authors	39%	65%	26%	61%	73%
TIMES Compression					
- Algol data	28.7	5.6	14.3	27.6	36.0
- Basic messages	2.2	0.6	1.3	1.1	4.5
- Decompression					
- Algol data	29.2	4.0	9.8	3.3	-
- Basic messages	1.9	0.5	1.0	0.3	-

Notes on Table 3

1, With the exception of Wagners algorithm all programs were written in Burroughs Extended Algol. Wagners algorithm is written in PL/1

2, The number of cards in the top three rows does not include comment cards.

3, The apparent discrepancy between the total number of cards and the sum rows 2 and 3 for each algorithm is due to overheads of timing, setting up dictionaries etc.

4, The "core code estimate" is that produced by the compiler.

5, The "best compression ratio" and times are also the only ones for the Huffman Coding, Pattern Substitution and Wagners algorithm; See tables 4, and 5, for other values of the compression ratio for different parameters for the Combining Character and Fixed Point Number Methods.

6, No expansion routine was written for Hahn's algorithm.

7, The formula for compression ratio is that given in section 3 i.e.

$$CR = \frac{100}{1} \quad X \quad \frac{\text{Output file (Bytes)}}{\text{Input file (Bytes)}}$$

8, The Algol source text file contained 531 records, each considered to contain 72 characters.

The Basic error messages file contained 46 records each containing 52 characters.

TABLE 4: Results For Different Values of the Parameters in the Combining Characters Algorithm.

No. Master Characters.	No. Combining Characters.	Compression Ratio.	Data.
6	32	58.48	
7	27	58.11	ALGOL
8	24	57.94	SOURCE
9	21	57.62	DATA
10	19	57.58	
11	17	57.80	
12	16	57.78	
6	37	56.23	
7	32	55.10	BASIC
8	28	54.01	ERROR
9	26	53.89	MESSAGERS
10	22	67.48??	
11	20	54.44	
12	18	54.85	
13	17	54.89	
14	16	99.62	
15	16	99.62	

Notes on Table 4.

1, Timings for decompression and compression were not significantly different and so were not included.

2, The anomaly in the data as indicated by ?? was not investigated.

TABLE 5: Results For Different Values of the Parameters in the Fixed Point Number Algorithm.

N= No of Symbols/word	B=size of dictionary	Compression Ratio	Data
7	47	39.92	ALGOL
8	29	39.17	SOURCE
9	20	39.92	TEXT
10	14	42.56	
11	11	44.07	
8	29	52.2	BASIC
9	20	54.2	ERROR
10	14	54.2	MESSAGES
11	11	54.2	
12	9	54.2	
13	7	60.2	

Notes on Table 5

Astimes for execution were very similar they are not included in the table.

Results for Pattern Substitution Expansion Algorithm Implemented on PDP-11.

Total of 46 records with 1155 characters originally. 8 common phrases used; total of 59 characters.

For overhead use.

3 bytes per common phrase

2 bytes per message.

Assume 40 bytes of code required for a message printing routine for uncompressed messages.

Size of messages + overheads = code for no phrase extraction
= 1155 + 46X2 +40
= 1287 bytes.

After phrases extracted (manually) message size = 834 bytes.
Code to restore and print messages = 110 bytes

Size of messages + phrases + overhead + code for phrase extraction.
= 834 + 59 + 46X2 + 8X3 + 110
= 1119 bytes

⇒ Nett saving of 168 bytes (13%)

⇒ Compression ratio of 87%

Results for Extended Coding Scheme Algorithm as Implemented on the PDP-11.

Code required 124 bytes

Tables required 35 bytes

Overheads required:
2 bytes per message.

Based on the 1st five error messages the compression ratio considering overheads was 54%

As the messages were originally 1155 characters:
54% of 1155 = 624 characters.

Add overheads:
624 + 2 X 46 = 716

Assume that as in the Pattern Substitution Algorithm 1287 bytes were required for an ordinary message printingscheme.

Extended Coding Scheme uses

716 + 124 + 35 = 875 bytes

⇒ *nett saving of 412 bytes (32%)*

⇒ *compression ratio of 68%*

6. DISCUSSION OF RESULTS.

6.1 Introduction

The results obtained from the main investigations of the project - as outlined by table 3 - can be seen to not always correspond to those results given by the people proposing the schemes. This discussion hopes to bring out the reasons for my results, for the differences between the results for different methods, and between my results and those of the other authors.

6.2 General

The first item to inspect is the method of calculation of the compression ratio- or rather the items that are included in it.

In my calculation of CR for the Huffman coding and Combining Character's methods no allowance for overheads on input or output was made. However, as variable length records were produced, there should have been an allowance of one word per record. In these two methods there was no special treatment of leading or trailing blanks, that is all characters on the input record were considered.

For the Fixed Point Number and the Pattern Substitution algorithm the compression ratio was calculated correctly. In these two methods all leading or trailing blanks are removed as part of the compression process.

Wagners PL/1 program includes three characters per record for both input and out put files. Leading and trailing blanks are not considered as part of the input records so records are of variable length.

This differing treatment of blanks is the explanation for the fact that for the Huffman Coding and Combining Characters algorithms the Basic error messages are compressed to a higher degree than the Algol text, while the opposite is true for the Fixed Point Number and Substitution algorithms.

The Basic error messages have a much lower proportion of short records than the Algol source text does and therefore fewer leading and trailing blanks. However the fewer number of different characters in the Basic error messages was undoubtly the reason for that file being compressed to a greater extent by the methods of Huffman coding and Combining Characters where no special treatment of blanks was employed.

6.3 Wagner's Algorithm.

Included in Table 3 are the results obtained for Wagner's algorithm. Not too much importance should be attached to these, as the algorithm, copied from an article (9), took some time to get working at all and is still not working as claimed.

6.4. Huffman Coding.

From the entropies (expected number of bits in a coded character) of the Huffman codes devised for the Algol source text and Basic error messages, it was expected that the compression ratio's would be 59% and 53 % respectively. It can be seen in Table 3 that the experimental results are both better by about 5%. This is probably due to my data not being a statistically large sample.

The Huffman code routine implemented is not as efficient in compressing the input file as the one written for the U.S. Navy(7). This is because the Navy system treated certain common patterns as single characters. It is felt that the times taken to compress and expand records could have been substantially reduced if more thought was given to the encoding of the algorithms. In particular a binary tree is perfectly suited to the requirements of the decompression routine and would appreciably reduce the number of comparisons and the amount of 'bit fiddling' required. Ruth and Kreutzer (7) also found that time for decompression was higher than time for compression and both were higher than they had wanted.

6.5. Combining Characters Algorithm.

The Combining Characters method can be seen to be very efficient in the use of computer resources even though the compression ratio for the method is not as high as for other methods. The reason for the low times and the small program are a very simple algorithm and only one test for each input and output character. It was possible to acheive a slightly better compression ratio than that quoted by Synderman and Hunt (8) because while they had both upper and lower case characters to deal with these tests used only upper case data.

No method was found to predict in advance of the tests which combination of master and combining characters would produce the best results. The results however show that for the Algol source text ten master and 18 combining characters achieve a marginally higher compression ratio than other combinations. For the Basic error messages 9 master and 26 combining characters seems best.

It can be seen that with this method's high speed it is particularly suited to the application Synderman and Hunt devised it for, namely an on-line, random access data entry and inquiry systems.

6.6. Fixed Point Number Algorithm.

As mentioned when introducing this algorithm in section 4 its efficency is greater if the full length of the computer word is used to represent integers. This possibly accounts for the fact that the compression ratios for my data are not as low as the figure of 26% Hahn quotes (3) as he implemented the algorithm on an IBM360.

Although the comparisions required on input are those necessary to find the characters location in a dictionary, and usually only one test is necessary on output, the times for compression and decompression are relatively high. This is due to the large number of multiplications and additions in compression, and divisions and substractions on expansion that are required. This must be considered a disadvantage when time is important.

The program testing the algorithm was run using different values of the parameters N (the number of symbols in a word) and B (the number of characters in the primary dictionary). This was done for both sets of data and the results can be seen in table 5. Using the frequency of occurence of each of the characters in the data sets it was possible to show that the N=8, B=29 combination for the Algol source text and N=10, B=14 combination for the Basic error messages would be the most efficient and this is reflected in the results. The suppression of leading and trailing blanks made it impossible to predict a value for the compression ratio however.

6.7. Pattern Substitution Algorithm.

The figure quoted for comparison purposes for the pattern substitution algorithm is that of Mayne and Jones (4), but as they do not 'squeeze off' leading and trailing blanks the comparison is probably misleading. However it can be seen that for a well chosen set of common phrases a substantial reduction in file size can be acheived by this relatively simple algorithm. A large number of comparisons results in a high compression time. Only 1 test per character is needed for expansion resulting in the fastest expansion routine of those tested.

6.8. Basic Error Messages on PDP-11

The pattern substitution algorithm was also implemented on the Computer Science Department's PDP-11 but only the decompression algorithm was programmed and compression was done 'manually'. As the results in the last section show this was worthwhile even with the size of program code taken into account.

This was arather surprising result as it was not expected that for the rather small set of Basic error messages there would be any advantage in using a text compression technique.

The results for pattern substitution were not as good as those for the extended coding scheme algorithm which produced an effective compression ratio of 68%.

For both the methods tested on the PDP-11 the teletype appeared to be printing at full speed although some computation was required in between printing characters.

6.9. *Summary*

In summary the results with the exception of those from the program of Wagner's are, in retrospect, what should have been expected.

7 SUMMARY.

It has been shown that there exists text compression techniques capable of producing a significant reduction in secondary memory requirements. The compression ratio figures can be made more interesting by looking at them in terms of dollars saved rather than percentages and along this line we can quote figures like $10,000 a month (7) clear savings because secondary memory expansion was avoided.

Most large text files are amenable to one or more compression techniques and the particular technique that best suits an application is the one that produces the greatest savings without degrading the system to an unacceptable extent.

Finding just what this technique is will probably remain a process of trial and error but this study should have shed some light on a few of the text compression methods possible.

8 REFERENCES

1, ALSBERG (1975) "Space and Time Savings Through Large Data Base Compression and Dynamic Restructing" Proc. IEEE, Vol 63. August 1875 Pages 1114-1122.

2, FAJMAN & BORGELT (1973) " The Wylbur Operating System." CACM. Vol 16, No 5, May 1973 Page 319 only.

3, HAHN (1974) " A New Technique for Compression and Storage of Data" CACM. Vol 17, No 8, August 1974 Pages 434-436.

4, MAYNE & JAMES (1973) "Information Compression by Factorising Common Strings" Computer Journal. Vol 18, No 2 Pages 157-160

5, REZA (1961) "Introduction to Information Theory." McGrow-Hill 1961 Chapter four only.

6, ROSEN et al (1965)

"The PUFFT System"
CACM. Vol 8, No 11 November 1965
Pages 665-666 only

7, RUTH & KREUTZER (1872)

"Data Compression for Large Business Files"
Datamation. September 1972
Pages 62-66

8, Synderman & Hunt (1970)

"The Myriad Virtues of Text Compaction"
Datamation. December 1970.
Pages 36-40

9, WAGNER (1973)

"Common Phrases and Minimum - Space Text Storage"
CACM. Vol 16. No 3. March 1973.
Pages 148-152.

"An Algorithm forExtracting Phrases in a Space-Optimal Fashion"
(Algorithm 444)
CACM. Vol 16. No 3 March 1973.
Pages 183-185.