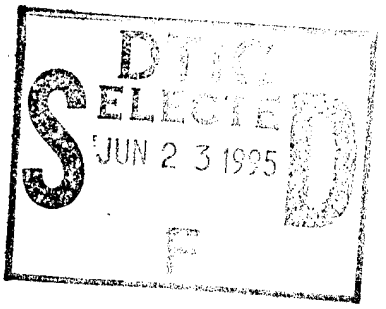
Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/8140.2--95-7742

Applications of Data Compression Techniques To A Communication System



JUNHO CHOI

Command Control Communications
Computers and Intelligence Branch
Space systems Development Department

MITCHELL R. GRUNES

Allied Signal Technical Services

June 21, 1995

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 21, 1995	

4. TITLE AND SUBTITLE

Applications of Data Compression Techniques to A Communication System

5. FUNDING NUMBERS

6. AUTHOR(S)

Junho Choi and Mitchell R. Grunes*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Space System Development Dept
Code 8000, C4I Branch
Naval Center for Space Technology

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/8140.2–95-7742

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

SPAWAR 40E
CMDR M. Hecker/Mike Regan
Code 451 NRL Code 9110
Washington, DC 20375-5320

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

*Allied Signal Technical Services

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A brief overview of current status of data compression technologies are presented with the interim development of compression software and tests of effectiveness on the selected digitized data types. Several algorithms such as Rice, LZ-family, and run-length codings are examined for selected digitized packet data streams. These algorithms are able to compress most data available with a fairly good compression factor except for Tape A and encrypted data. Tape A data contains a mixture of several data formats in very short runs (several packets), rather than the long runs (typically hundreds of packets), for which the current version of the software was designed. the remaining data sets were compressed with a compression factor of over 2.74.

14. SUBJECT TERMS

Constant coding	DPCM	Run-length coding	LZRW3A
Constant bit removal	Rice coding	Huffman coding	LZ77
Factor removal	Lempel-ziv coding	Dictionary search	LZ78

15. NUMBER OF PAGES

22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18
298-102

TABLE OF CONTENTS

APPLICATIONS OF DATA COMPRESSION TECHNIQUES TO A COMMUNICATION SYSTEM

1. INTRODUCTION

1.1 Background

There has always been an interest in economical communication, whether it be oral, written, electromagnetic, or digital. There is still today a broad use of abbreviations and acronyms in both oral and written material. It seems that text compression comes naturally to people designing a code for communication. When Samuel Morse was contemplating codes for an electromagnetic telegraph in 1832, he initially considered schemes where only the 10 numerals could be transmitted. He would then use a codebook for words, names, dates, and sentences. By 1835 he had abandoned this idea in favor of the celebrated Morse code, which uses dots and dashes to represent letters, numerals, and punctuation. The Morse code, which speeded up telegraphy, is an example of an early data compression technique. In 1939 Dudley invented the VOCODER (VOICE CODER), which made the transmission of voice possible over a very narrow telephone channel bandwidth. Digital communication is now replacing almost all forms of analog communication, and the need for digital data compression is growing[1].

Data compression is the reduction in the amount of signal space that must be allocated to a given message set or data sample set. This signal space may be in a physical volume, such as a data storage medium like magnetic tape; an interval of time, such as the time required to transmit a given message set; or in a portion of the electromagnetic spectrum, such as the bandwidth required to transmit the given message set. All these forms of the signal space--volume, time, and bandwidth--are interrelated. Thus, a reduction in volume can be translated into a reduction in transmission time or bandwidth. The parameter to be reduced or compressed usually determines where the data compression operation will be performed in the system. Throughput rates required can vary enormously depending on the application (e.g., consider an inexpensive serial modem for telephone lines operating at 9600 bit-per-second (bps) versus digitized high definition video on an optical link that may require over one billion bps in uncompressed form).

For both storage and communications applications, the trade-off between the amount of compression achieved by a given algorithm and how fast and inexpensively it can be implemented in software or hardware can be a key issue. Another key issue is the ability of a given algorithm to adapt to different types of data, because in many applications there may be no advance knowledge of what type of data will be present. Data compression techniques can be effectively applied to diverse data types, including written natural language text, computer source and object code, bit-maps, numerical data, graphics, CAD data, map and terrain data, speech, music, scientific and instrument data, fax and half-tone data, gray-scale and color images, medical data and imagery, video, animation, and space data [2].

The relative importance of these parameters has varied over the years. Initially there was--and still is--a great deal of interest in reducing the bandwidth

Manuscript approved December 6, 1994.

required to transmit analog signals, as in telephony and television. Later, in systems such as facsimile, increasing the speed of transmission became important. Finally, today, the volume of the data is the critical parameter in need of reduction in many systems, especially space--ground communication systems. Data compression has been called other names in the literature. The two most common names are data compaction (or companding) and source coding.

The first essential signal processing step in digital communication systems, formatting, makes the source signal compatible with digital processing. Transmit formatting is a transformation from source information to digital symbols (in the receive chain, formatting is the reverse transformation). When there is data redundancy reduction or data compression, in addition to formatting, the process is termed source coding. The goal of source coding is either to improve the signal-to-noise ratio (SNR) for a given bit rate or to reduce the bit rate for a given SNR. Several techniques used in source coding are: amplitude quantizing (uniform or non-uniform), differential pulse code modulation (DPCM), one-tap or N-tap and adaptive coding, block coding (vector quantizing, transform coding, subband coding), synthesis and analysis coding (vocoders and linear predictive coding--LPC), redundancy reduction coding (Huffman coding, run-length coding, arithmetic coding, Rice coding, Lempel-Ziv coding etc.) [3].

For communication over a digital communication link or storage in digital memory, digital data compression is the conversion of a stream of high-rate data into a stream of relatively low-rate quantized data. The goal is to reduce the volume of data transmitted over a digital channel or a digital medium. As the volume of text, speech, map or terrain data, space data, image and video data becomes prohibitively large in the near future for many communication links or storage devices, the theory and practice of data compression are receiving increased attention. Some areas have already faced the limit of transmission bandwidth or storage, such as remote sensing or global environmental observation and multimedia technology.

Many data sources contain significant redundancy--symbol distribution, pattern repetition, and positional redundancy and correlation. Data compression algorithms can be broadly categorized into two classes: lossless (or redundancy reduction or reversible) and lossy (or entropy reduction or irreversible). They can be also classified as static (or fixed) and dynamic (or adaptive). Often, in attempting to classify a variety of techniques into only two classes, some techniques will fall into both classes. This is the case with the classification of data compression techniques. Although no one classification approach is perfect, lossy versus lossless labels are used in Table 1 [4, 7, 8].

Lossy data compression concedes a certain loss of accuracy in exchange for increased compression. Lossy compression has often proved to be effective when applied to graphics, images, video, satellite data, speech and digitized voice. These digitized representations of analog phenomena are not perfect to begin with, so the idea of output and input not matching exactly is a little more acceptable. Most lossy compression techniques can be adjusted to different quality levels, gaining higher accuracy in exchange for less effective compression.

Table 1 Classification of Data Compression Techniques

Lossy			Lossless		
Transform	Search	Predictive	Optimum Source	Predictive	Others
K-L	code book	DPCM	Huffman	Predictors	run-length
DCT	tree	DM	Shannon-Fano	Interpolator	quad-tree
Hadamard	sequential	AR models	Elias-arithmetic	Q-coder	bit-plane
VQ	LZ family	Markov-mesh	Rice	LZ family	hybrid
SVD		LPC			nonstandard
Fractals					scanning
Principal component					progressive-
Fourier					schemes
Haar					

Notes: Some of the techniques are used in both lossless and lossy compression and most of the real applications are hybrids of several of the algorithms, as in the software developed for this report.

Lossless data compression consists of those techniques guaranteed to generate an exact duplicate of the input data stream after a compress/expand cycle. This type of compression is generally used when storing database records, spreadsheets, word processing files, electronic mail, etc., and also during transmission/reception of telemetry and command data in space-ground communications systems. In these applications, the loss or modification of even a single bit is often catastrophic [5].

1.2 OBJECTIVES

The main purpose of this report is to present a general overview of status of data compression technologies and to present the interim development of our compression software, including our tests of effectiveness on the selected data types currently available. With the explosive growth of modern computer and communications technology, the demand for data transmission and storage has increased dramatically from a few kilobits per second (Kbps) to a few gigabits per second (Gbps), especially in satellite communications. Before the end of this century, this demand will reach over 10 Gbps [6]. To meet this requirement and future challenges, advances in the technology for mass storage, data communications, and data processing must be occur.

Data compression can be useful for various data processing applications. Computer networks require data to be transmitted from one site to another. Data compression can reduce communication costs in computer networks by compacting messages before transmission. Data compression can also reduce the storage requirements of databases and file systems, and thereby increase the effective capacity of storage systems. The usefulness and necessity of data compression arise in data storage and transmission bandwidth. One way to meet these demands is to implement advanced data compression techniques to improve the efficiency of data

transmission and storage. Note, also, that the effectiveness of a compression algorithm is measured by its data compressing ability, the resulting distortion, and by its implementation complexity. The major objective of this task is to develop efficient data compression algorithms to meet current needs as well as future challenges. In section II, systems design considerations are discussed. A brief overview of selected compression algorithms is presented in section III. Compression test results are presented in tabular form in section IV, along with comments. Finally a conclusion and some recommendations are presented in section V, and references are located in section VI.

2. SYSTEMS DESIGN CONSIDERATIONS

An effective data compression routine or series of compression algorithms requires an examination of the overall transmission system, and an analysis and understanding of the composition of the data to be transferred. An analysis of the complete transmission system is a prerequisite to maximizing the efficiency of communication systems. This analysis should examine not only hardware and software, but must also consider the volume and type of data traffic expected to be communicated. For example, consider a remote batch computer interfaced to a magnetic tape unit and to a communications line, functioning as a stand-alone tape-to-tape transmission system. We can break up this system into its logical components: the magnetic tape subsystem, the remote batch computer and the communications link.

In regards to the communications link, the protocol employed and the modem's data transfer rates govern the line utilization efficiency. If a higher speed modem is employed or a more efficient transmission protocol utilized, an increase in the number of characters transmitted per unit time can be expected. The remote batch computer itself can affect the overall transmission efficiency. If we consider the remote batch computer as a black box, it can accept input from the magnetic tape unit at a certain data transfer rate, process the data and then transfer the data to the communications line at another data transfer rate. The data transfer rate to the communications line will depend upon the channel adapter connecting the terminal to the modem, the modem's data transfer rate and the transmission medium employed. In addition, the processing performed by the computer will determine whether the device can output data fast enough to use the communications facilities at their rated data transfer rate or at some average rate below the rated level[3].

The unique characteristics of compressed data have important implications to the design of space science data systems, science applications, communications and data compression techniques. The sequential nature and data dependence between each of the sample values within a block of compressed data introduces an error multiplication/propagation factor that compounds the effects of communication errors. The data communication characteristics of the on-board data acquisition, storage and telecommunication channels may influence the size of the compressed blocks and the frequency of included reinitialization points.

The organization (size and structure) of the source data is continually changing, as is the entropy. This results in a variable output rate from the system that may require buffering to interface with the spacecraft data systems. On the ground, there exist key trade-off issues associated with the distribution and management of the science and communications data products when data

compression techniques are applied to alleviate the constraints imposed by ground communication bandwidth and data storage capacity. Missions that anticipate using data compression could improve their information throughput efficiency by influencing sensor and instrument design to be synergistic with spacecraft data acquisition and data management schemes, science application requirements (including quick look data analysis), and characteristics of the data collection and downlink communication channels. In other words, the theory of data compression, along with its application and design effects, must be understood in the context of the end-to-end information system.

The incorporation of data compression techniques into a larger data system (or existing computer systems) has to be carefully planned, with a number of trade-offs considered if the compression operation is to obtain the maximum benefit. Before one can select a data compression technique that will meet the system requirements of the specific application, one should examine necessary properties for data compression such as good compressibility, fast decodability, random accessibility, timing and buffering between compressor and transmission channel, inherent distortion, error effects, and error control [1, 2, 9]. Additional system considerations when integrating compression techniques into an existing computer system should include but are not limited to:

1. The compressed message length is unpredictable because it depends on the content of the input message.
2. Error tolerance: a given error rate may be acceptable for the transmission of uncompressed data, but the same rate may be unacceptable for compressed data. Error correction coding is strongly recommended.
3. Block size: short blocks are penalized by the start-up overhead needed to convey subject statistics and for synchronizing and error control. For example, Huffman coding must send a translation table or equivalent information with the message to give the encoding for each character. Lempel-Ziv methods are inefficient in early sections of the message, until character strings are encountered that repeat earlier strings. Large blocks suffer a loss in efficiency because the block may lack stable statistics. This is a typical occurrence in commercial computer data. For example, program development files may contain intermixed blocks of source code, relocatable object code, and executable modules of machine code. Other digitized sample data typically has regions of high and low activity
4. Location of the compression: When using compression on computer peripherals, a significant system consideration occurs in deciding whether to implement compression in the input/output (I/O) path in the device controller, the channel, or the central processor.
5. Disk storage.
6. Tape storage: record length, density and blocking factor.
7. Communications: Facsimile.
8. System-wide application.

9. Software linkage considerations:
   a) the type of device the software will operate on;
   b) the method used to link the compression software to other software;
   c) the transfer rate of the compressed data--either internally to or from peripheral storage units, or to and from a transmission medium;
   d) the number and complexity of instructions required for coding t the appropriate software--single or double word, memory references, number of shifts, and arithmetic complexity;
   e) the processor timing required for decompression;
   f) the flexibility of the compression algorithms.

Based upon these considerations and requirements, current efforts on this task have been constrained to lossless compression schemes even though both lossy and lossless techniques may be eventually implemented. Most of the data currently processed can be characterized as either digitized data, time codes, or flag bits of various types. This may explain the reason that the dictionary search compression algorithms (i.e., Lempel-Ziv families), which were primarily designed to work with natural language text information, did not work very well on most of our data. Although the software is expected to apply to specific data formats, the compression software is written as generally as practical, to allow for adaptation to other purposes. Furthermore, it is required that the result of losslessly compressing and decompressing any data stream should produce a result identical to the original, even if the input format does not match the assumed format. An average compression factor of at least two is considered absolutely necessary.

3. OVERVIEW OF SELECTED LOSSLESS COMPRESSION ALGORITHMS

As mentioned above, only lossless compression algorithms were tested from all three categories of table 1--optimum source coding (Rice coding), dictionary search coding (L-Z family), other coding techniques (constant bit removal, subtraction of the minimum value, common factor removal, run-length coding, and constant coding), and various combinations of those techniques. Each technique will be briefly reviewed with its advantages and disadvantages. As pointed out in the previous section, lossless techniques remove or reduce that portion of the data which can be reinserted or reconstituted at the receiving end of the system with no residual distortion.

3.1 RICE CODING

Rice coding employs a "split-coder". The last significant fraction is transmitted uncompressed, but the most significant fraction is compressed. Rice coding is similar to Golomb coding except that only a subset of the parameter values may be used, namely the powers of 2. The Rice code with parameter k is exactly the same as the Golomb code with parameter $m = 2^k$. Hence to encode an integer n using the Rice code with parameter k, we first compute $[n/2^k]$ and output this integer using a unary code. Then we compute $(n \bmod 2^k)$ and output this value using a k-bit binary

code. The resulting codes give somewhat less compression efficiency than Golomb codes, however, they are even easier and faster to implement than Golomb coding, especially in hardware, since we can compute $[n/2^k]$ by shifting n right by k bits, and compute $(n \bmod 2^k)$ by masking out all but the k lowest order bits of n[11]. In addition, there is less overhead, which allows faster adaptivity.

A common method of obtaining and transmitting coding parameters in algorithms that use Rice coding is to divide the data into blocks, and for each block to estimate the code lengths that would be obtained using each of a set of reasonable parameter values, and output the best parameter value as side information. Rice coding is similar to Huffman coding in the sense of remapping bit width. In detail, Rice coding uses a Laplacian frequency model to reduce the number of bits needed to specify the remapping. Except for this reduced overhead, Rice coding is identical to Huffman coding, as long as the Laplacian model applies. This model is a fairly good approximation of the distribution of the remapped first differences. The reduced number of adaptive bits allows very rapid adaptivity.

Rice coding can be used as an alternative to arithmetic coding or Huffman coding in almost any setting requiring adaptive modeling. All that is required is that the events to be encoded be arranged in approximately descending order of probability. In image and other digitized sample data, after first differencing, the ordering follows simple patterns such as (0,1,-1,2,-2,...). In the other applications it is possible to maintain approximate ordering by using heuristics such as move-to-front (move an event to the head of the list whenever it occurs) or transpose (move an event up one place in the list whenever it occurs). In all the mentioned cases the parameter values are the only overhead that is required.

Rice coding gives even faster coding than Huffman coding because of the especially simple prefix codes involved. Furthermore, adaptive modeling is possible without the complicated data structure manipulations required in dynamic Huffman coding. The main drawback to Rice coding is the limited class of distributions that can be modeled exactly. But, this is not a serious problem (unless one event's probability is close to 1) because the probabilities of the more probable events will be estimated fairly well.

3.2 LEMPEL-ZIV (or ZIV-LEMPEL) CODING

Most general compression schemes used statistical modeling until Lempel and Ziv introduced an adaptive dictionary-based compression algorithms. Lempel-Ziv (will be called LZ coding hereafter) coding refers to two distinct but related families of coding techniques first presented by Lempel and Ziv in two papers published in 1977 (LZ77) [13] and 1978 (LZ78) [14]. The fundamental idea behind LZ77 is that substrings of the message are replaced by a reference (e.g., offset and length) to a substring in an earlier part of the message. LZ77 is a relatively simple family of algorithms. The dictionary consists of all the strings in a window into the previously read input stream. A file-compression algorithm, for example, could use a 4 Kbyte window as a dictionary. In our case, the dictionary consists of the previous symbols of the current block. While new groups of symbols are being read in, the algorithm looks for matches with strings found in the previous window of data already read in. Any matches are encoded as references sent to the output stream. LZ77 and its variants make attractive compression algorithms. Some off-the-shelf programs,

such as LHarc, use variants of LZ77, and have proven to be somewhat popular. But they have been largely replaced by LZ78, which is faster and may be better in most cases.

The LZ78 family takes a different approach to building and maintaining the dictionary. Instead of having a limited-size window into the preceding text, LZ78 builds its dictionary out of the previously coded strings of symbols in the input text. LZ78 has achieved more popular success than LZ77, due to the LZW adaptations by Welch [15] and others, which form the core of programs such as PKZIP and UNIX Compress. These two algorithms have sparked a flood of variants that use dictionary-based methods to perform compression, some of which can be implemented at very high speeds. Some representatives of LZ codings are listed in Table 2 by chronological order [9, 17]. We selected LZRW3-A as our test algorithm.

Table 2 Principal LZ Algorithms and Their Characteristics

LZ77	Ziv and Lempel (1977)	Pointers and characters alternate. Pointers indicate a substring in the previous N characters.
LZ78	Ziv and Lempel (1978)	Pointers indicate a previously parsed substring.
LZR	Rodeh et al (1981)	Pointers indicate a substring anywhere in the previous characters.
LZW	Welch (1984)	The output contains pointers only. Pointers indicate a previously parsed substring, and are of fixed size.
LZMW	Miller and Wegman(1984)	Same as LZT but phrases are built by concatenating the previous two phrases.
LZJ	Jakobson (1985)	The output contains pointers only. Pointers indicate a substring anywhere in the previous characters.
LZC	Thomas et al (1985)	The output contains pointers only. Pointers indicate a previously parsed substring.
LZSS	Bell (1986)	Pointers and characters are distinguished by a flag bit. Pointers indicate a substring in the previous N characters.
LZB	Bell (1987)	Same as LZSS, except a different coding is used for pointers.
LZH	Brent (1987)	Same as LZSS, except Huffman coding is used for pointers on a second pass.
LZT	Tischer (1987)	Same as LZC but with phrases in a LRU list.
LZFG	Fiala and Greene (1989)	Pointer select a node in a tree. Strings in the tree are from a sliding window.
LZRW	Ross Williams (1991)	Same as LZ77 except for a speed. Faster and simpler LZ algorithm.

3.3 RUN LENGTH CODING

Run length coding (sometimes called run-length encoding, RLE) is a technique that parses the message into consecutive sequences (runs) of identical instances. As with Huffman coding, run length coding takes many forms and has been used as a component of many compression algorithms. Run-length coding can

usually be identified by its trade mark of coding a run of identical data values by a single instance of the repeated value together with a repetition count [1, 2, 8].

Run-length coding works when symbols do not occur independently but are influenced by their predecessors. Given that a symbol has occurred, that symbol is more likely than others to occur next. If this is not the case, coding runs (rather than symbols) will not compress the information. The same effect can be achieved in a more general way by other coding techniques, but run-length coding uses very little overhead when the runs are long.

3.4 OTHER UNCONVENTIONAL CODING

In addition to those three lossless techniques, we have used constant coding, constant bit removal, adaptive first differencing, and factor removal. Other combinations, such as hybrids of Rice and run-length coding were tried in an attempt to improve compression factors. The details of the applied methodology are described in the following section.

4. COMPRESSION TEST RESULTS

Several assumptions and basic requirements have been made to develop compression algorithms such as:

1. Most of data are characterized as digitized data, time codes, or flag bits of various types.
2. An average compression factor of at least two is considered absolutely necessary, but more is desirable. This emphasis causes this task to differ substantially from other compression tasks, where execution speed tends to be more important than compression factor.
3. The software is being developed in as general form as practical, to allow for easy adaptation to other purposes. However, our application will mostly work on specific known data formats. Adaptation of the software to the specific formats, including specification of bit field boundaries, has a strong impact on compression factor, and should be done whenever possible.
4. It is required that the result of losslessly compressing and decompressing (or reconstructing) any data stream should produce a result identical to the original, even if the input data formats do not match the assumed format.

4.1 TEST APPROACHES

Our software was evaluated with seven different data sets: Tape A, Tape B, Tape C, Tape D, Tape E, Tape AA, and Tape BB. The formats of data included (except for Tape BB) are described in [19]. These data sets include normal mode commands, block mode data from commlink, GMT2 data packets, range, Ephemeris and GMT (REG) history packets, digitizer subsystem source, on-board processor (OBP) subsytem source, narrow band downlink formatter subsystem source, etc.

Unix Compress and PKZIP were tested on Tape AA. However, these did not yield very high compression factors: around 1.8.

In our work, the data is broken up into blocks of input packets, each of which is compressed into a single compressed packet. Two factors influence the block boundaries:

1. A maximum number of input packets (512) is used to reduce problems due to transmission errors and processing delays.
2. The packets are classified into four types. When a new type occurs, a new block is always started. This ensures more homogeneous statistics.

Each bit field is processed separately. Many different compression schemes are tried on each field, within each block. The one that works best is used. Although this is inefficient in terms of speed, it should produce relatively high compression factors. The compression algorithms include: no compression, constant coding, constant bit removal, run-length coding, adaptive first differencing and factor removal, Rice coding, Rice coding with run-length coding, LZ77, LZ77 on remapped differences, LZRW3-A, and LZRW3-A on remapped differences. A brief discussion of each algorithm is presented below.

No Compression: When none of the compression methods perform any actual compression, the code indicating that the sequence is uncompressed is sent, and the data is actually sent unchanged.

Constant Coding: Many of the fields were constant within any given block of data. The value is sent only once for the whole block. For those fields and blocks where it works, this produces a very high compression factor.

Constant Bit Removal: In most of the data fields that were not completely constant, some bits were constant within any given block of data. A mask is sent by specifying which bits are constant, and their values are also sent. The remaining bits are sent unchanged. This method was fairly successful in many instances. Therefore, constant bit removal is also applied whenever possible before all the remaining algorithms.

Run-Length Coding: Data, after constant bit removal, was reorganized and transmitted as ordered pairs, containing the data value, and the number of times the value was repeated. The number of bits needed to specify the largest repeat count was determined, and sent before all of the ordered pairs.

Adaptive First Differencing And Factor Removal: In many cases the values, after constant bit removal, can be transformed into smaller numbers by first differencing. This leads to a better compression factor. A number of variations are tried as listed below, to provide the best possible results.

1. The bit field is adaptively interpreted as unsigned, signed (two's complement), and the least significant fraction of a larger number. Adaptation here has been performed by minimizing the sum of the remapped differences.
2. The values are remapped into positive numbers using an adaptive scheme in which it is tested whether positive or negative first differences yield a smaller remapped sum.
3. Prior to remapping, the minimum value is subtracted off, and the maximum value is determined, and the differences are tested to determine whether all of the differences are of one sign, in order to provide optimal results.
4. The possibility is tested that the undifferenced data works better than remapped differences.
5. The greatest common factor is removed at two points in the process.

The results are not sent directly. Instead, the remapped first differences are used in some of the remaining compression methods.

Rice Coding: The remapped differences are split into a most significant fraction, and a least significant fraction. The most significant fraction is sent as a terminated base 1 number (0 is sent as 1, 1 is sent as 01, 2 is sent as 001, ...), and the least significant fraction is sent unchanged, on the assumption that it is essentially random. The optimal number of bits in the least significant fraction (an integral number) is found from an approximate formula based on the sum of the remapped differences.

Rice Coding With Run-Length Coding: An attempt was made to combine Rice coding with run-length coding. Several approaches were tried, including:

a. The value and repeat counts from the run-length coding were treated entirely separately through Rice coding. This didn't perform very well, and has been abandoned. However, it is possible that a variant of Rice coding may cause it to work better in the future.
b. Within each Rice block, long runs of zero were assigned special codes. Several schemes were tried without much success.
c. One very simple scheme, in which the highest probability symbol, sent in a single bit, represents a run length code of four zero-valued differences, works best in the lowest entropy cases. In addition, a separate Rice adaptive code is used for the case where all the remapped differences in the Rice block are zero. Since these methods can be added at little cost, our current Rice coding software always tries them in low entropy cases.

LZ77: A simple dictionary search scheme based on LZ77 was applied to the data, after the constant bit removal process. This attempts to match strings of source symbols to prior strings, and then transmits pointers and lengths of the matches. The results are not very good on most of our test data, but worked well on a file containing text strings. Note that LZ77 and LZ78 were designed to be used with natural language text, which is not the majority of what is being compressed in this study.

LZ77 On Remapped Differences: The LZ77 method was applied to the remapped differences. This hybrid method sometimes did better than LZ77 alone.

LZRW-3A: This version is a more modern and faster dictionary search scheme, based largely on LZ78, designed by Williams [17]. (We have actually used an improved version which Williams has not published.) For most of our test data, it did not do very well, probably for the same reasons as LZ77.

LZRW-3A on Remapped Differences: LZRW3-A was applied to the remapped differencing method. As with LZ77, this hybrid method sometimes performed better than LZRW3-A alone.

4.2 COMPRESSION RESULTS

The input data whose results are presented in tables 1,2,3 and 5 include range, Ephemeris, GMT, digitizer, range and synchronizer, on-board processor, and header information data [19]. Table 4 presents results from a data type which we have not yet identified, but which includes some text. Table 6 presents results from some specially processed type of data, and table 7 presents the results from encrypted data.

Runs were made with our prototype software on these seven sample data sets, which we shall call A, B, C, D, E, AA and BB. The sample data sets contained packets which were automatically classified to be one of 3 types:

1. Data lacking the "sync bits" corresponding to the specified packet format. Type one packets are simply treated as a string of bytes, without regards to the "correct" bit field boundaries.
2. Data having the correct sync bits, but not one of those data formats which are expected to have long continuous runs. A somewhat generic breakdown of bit fields is employed.
3. A data format (digitizer data) expected to have many long continuous runs. The expected breakdown of bit fields is employed.

The tables in this appendix will use the following notations:

Notation	Definition
Field #	Bit Field number
Width	Width of each field in bits
Meth 1 CF	Compression factor from constant coding
Meth 2 CF	Compression factor from constant bit removal
Meth 3 CF	Compression factor from run length encoding
Meth 4 CF	Compression factor from Rice coding
Meth 5 CF	Compression factor from LZ77
Meth 6 CF	Compression factor from LZ77 on remapped differences
Meth 7 CF	Compression factor from LZRW3-A
Meth 8 CF	Compression factor from LZRW3-A on remapped differences
Best Meth	The method number that did best on that field
CF w/OH	The compression factor from that method, *when extra overhead identifying the compression method is included.*
n/a	Not applicable. This includes the following cases: Meth 1: Values are not constant. Meth 2: No bits are constant. Meth 5,6: More than 128 distinct values (would be very slow). Meth 7,8: Reduced field width is over 8 bits, or there are more than 4096 values to be compressed.

Table 1: Compression Results for dataset of Tape A

A 5000 packet sample dataset which contains type 2 and type 3 packets. This contains very short runs (typically several packets) of each packet type. Consequently it could not be compressed very well with our current software, which includes significant start-up overhead for each run. We could do somewhat better if we reduced our overhead. Our software achieved a net compression factor of 1.23.

A typical run of 4 type 2 packets produced the following statistics:

Field #	Width	Meth 1 CF	Meth 2 CF	Meth 3 CF	Meth 4 CF	Meth 5 CF	Meth 6 CF	Meth 7 CF	Meth 8 CF	Best Meth	CF w/OH
1	2	4.00	2.00	0.89	1.33	0.80	0.80	0.12	0.13	1	1.00
2	1	4.00	2.00	0.57	1.00	0.50	0.50	0.06	0.07	1	0.57
3	4	n/a	n/a	0.67	0.57	0.70	0.55	0.24	0.20	0	0.73
4	9	n/a	1.09	1.00	0.97	1.12	0.92	0.47	0.41	5	0.95
5	10	4.00	2.00	1.60	1.82	1.54	1.54	0.48	0.53	1	2.50
6	16	n/a	1.45	1.33	1.42	1.42	1.36	0.70	0.67	2	1.28
7	16	n/a	1.08	1.10	1.42	1.28	1.36	n/a	0.67	4	1.25
8	6	n/a	1.00	0.86	0.80	0.96	0.75	0.33	0.30	2	0.80
9	8	n/a	n/a	0.88	0.96	1.16	0.89	0.86	0.80	5	1.13

Note that Field 9 contains 8 values per packet.

A typical run of 6 type 3 packets produced the following statistics:

Field #	Width	Meth 1 CF	Meth 2 CF	Meth 3 CF	Meth 4 CF	Meth 5 CF	Meth 6 CF	Meth 7 CF	Meth 8 CF	Best Meth	CF w/OH
1	2	6.00	3.00	1.33	2.00	0.75	1.20	0.14	0.16	1	1.50
2	1	6.00	3.00	0.86	1.50	0.43	0.75	0.07	0.08	1	0.86
3	4	n/a	n/a	0.67	0.75	0.83	0.71	0.29	0.26	0	0.80
4	9	n/a	2.35	2.16	1.64	1.64	1.50	0.56	0.55	2	1.86
5	10	6.00	3.00	2.40	2.73	1.88	2.31	0.60	0.65	1	3.75
6	16	6.00	3.00	2.59	2.82	2.18	2.53	0.86	0.92	1	4.36
7	16	6.00	3.00	2.59	2.82	2.18	2.53	0.86	0.92	1	4.36
8	3	6.00	3.00	1.64	2.25	1.00	1.50	0.21	0.23	1	2.00
9	3	6.00	3.00	1.64	2.25	1.00	1.50	0.21	0.23	1	2.00
10	3	n/a	1.64	1.38	0.86	0.86	0.75	0.21	0.21	2	1.06
11	2	n/a	1.33	1.09	0.63	0.63	0.55	0.14	0.14	2	0.80
12	11	n/a	1.06	1.43	1.12	1.32	0.96	0.70	0.55	3	1.27
13	2	6.00	3.00	1.33	2.00	0.75	1.20	0.14	0.16	1	1.50
14	6	n/a	1.33	1.16	1.03	1.12	0.84	0.40	0.38	2	1.09
15	3	6.00	3.00	1.64	2.25	1.00	1.50	0.21	0.23	1	2.00
16	5	n/a	1.50	1.67	1.03	1.15	0.94	0.34	0.32	3	1.25
17	2	6.00	3.00	1.33	2.00	0.75	1.20	0.14	0.16	1	1.50
18	14	6.00	3.00	2.55	2.80	2.10	2.47	0.78	0.84	1	4.20
19	16	n/a	1.68	1.68	1.33	1.75	1.20	0.90	0.74	5	1.57

Table 2: Compression Results for Dataset from Tape B

A 5000 packet sample dataset which contains type 3 packets with a few isolated type 2 packets. Our software achieved a net compression factor of 2.88.

A typical run of 512 type 3 packets produced the following statistics:

Field #	Width	Meth 1 CF	Meth 2 CF	Meth 3 CF	Meth 4 CF	Meth 5 CF	Meth 6 CF	Meth 7 CF	Meth 8 CF	Best Meth	CF w/OH
1	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
2	1	512.00	256.00	32.00	28.44	5.57	5.57	0.86	0.87	1	73.14
3	4	512.00	256.00	93.09	85.33	20.90	20.90	3.41	3.46	1	204.80
4	9	n/a	1.79	1.68	2.15	2.00	1.51	1.73	1.46	4	2.14
5	10	n/a	9.64	119.07	76.42	44.14	37.93	7.59	9.49	3	104.49
6	16	n/a	1.99	103.70	86.23	55.35	56.11	11.51	11.49	3	96.38
7	16	n/a	1.14	1.28	3.70	n/a	2.22	n/a	n/a	4	3.69
8	3	512.00	256.00	76.80	69.82	16.00	16.00	2.57	2.60	1	170.67
9	3	n/a	2.97	52.97	28.98	15.06	12.69	2.32	2.29	3	43.89
10	3	n/a	1.49	3.86	6.65	4.45	4.11	1.80	1.77	4	6.48
11	2	n/a	1.99	7.47	10.04	5.95	5.36	1.50	1.46	4	9.48
12	11	n/a	n/a	0.82	1.08	n/a	n/a	n/a	n/a	4	1.08
13	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
14	6	n/a	n/a	0.74	1.13	0.57	0.49	0.74	0.74	4	1.12
15	3	n/a	1.49	1.86	3.33	2.28	2.24	1.40	1.38	4	3.29
16	5	n/a	n/a	0.75	1.11	0.63	0.63	0.68	0.68	4	1.11
17	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
18	14	512.00	256.00	170.67	162.91	60.75	60.75	11.56	11.71	1	358.40
19	16	n/a	1.06	1.00	1.05	n/a	n/a	n/a	n/a	2	1.06

Table 3: Compression Results for Dataset from Tape C

A 5000 packets sample dataset, which contains type 3 packets with a few isolated type 1 and 2 packets. Our software achieved a net compression factor of 2.74.

A typical run of 512 type 3 packets produced the following statistics:

Field #	Width	Meth 1 CF	Meth 2 CF	Meth 3 CF	Meth 4 CF	Meth 5 CF	Meth 6 CF	Meth 7 CF	Meth 8 CF	Best Meth	CF w/OH
1	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
2	1	512.00	256.00	32.00	28.44	5.57	5.57	0.86	0.87	1	73.14
3	4	n/a	n/a	0.84	2.35	1.21	1.20	0.88	0.86	4	2.34
4	9	n/a	1.79	1.44	2.00	2.02	1.54	1.64	1.43	5	2.01
5	10	512.00	256.00	150.59	142.22	46.55	46.55	8.37	8.48	1	320.00
6	16	512.00	256.00	178.09	170.67	67.15	67.15	13.13	13.30	1	372.36
7	16	n/a	1.45	1.46	5.11	n/a	2.53	n/a	2.32	4	5.09
8	3	512.00	256.00	76.80	69.82	16.00	16.00	2.57	2.60	1	170.67
9	3	512.00	256.00	76.80	69.82	16.00	16.00	2.57	2.60	1	170.67
10	3	n/a	n/a	11.05	8.68	7.68	5.91	2.33	2.23	3	10.59
11	2	n/a	1.99	17.96	14.22	8.53	7.16	1.52	1.50	3	16.25
12	11	n/a	n/a	0.88	1.09	0.87	n/a	n/a	n/a	4	1.09
13	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
14	6	n/a	n/a	0.82	1.18	0.59	0.51	0.74	0.74	4	1.17
15	3	n/a	1.49	1.83	2.94	2.34	2.30	1.32	1.30	4	2.91
16	5	n/a	n/a	0.72	1.08	0.69	0.54	0.71	0.62	4	1.08
17	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
18	14	512.00	256.00	170.67	162.91	60.75	60.75	11.56	11.71	1	358.40
19	16	n/a	n/a	0.94	0.98	n/a	n/a	n/a	n/a	none	1.00

Table 4: Compression Results for Dataset from Tape D

This was a 5000 packet sample data set given to us that did not meet the specified data format. Consequently, it was all classified as packet type 1, and compressed purely as a stream of bytes, without regards to actual bit field boundaries. In spite of this, our software achieved a net compression factor of 7.69, indicating that the input data stream is very repetitive. The relatively good performance of Methods 5 and 7 probably also reflects one of the prime advantages of dictionary search compression techniques: although they rarely provide results that are at all close to optimum, they can provide fairly good results even when the precise format of the data is unknown, because there is a tendency to have repeated sequences of bytes in almost any format. This is undoubtedly one of the reasons that the most widely used generic compression software, such as Unix "Compress" and PKWARE "PKZIP", all use dictionary search techniques.

A typical run of 512 type 1 packets produced the following statistics:

Field #	Width	Meth 1 CF	Meth 2 CF	Meth 3 CF	Meth 4 CF	Meth 5 CF	Meth 6 CF	Meth 7 CF	Meth 8 CF	Best Meth	CF w/OH
1	8	n/a	1.14	1.45	2.38	7.57	6.17	n/a	n/a	5	7.57

Note that Field 1 contains 16 values per packet.

Table 5. Compression Results for Dataset from Tape E

A 5000 packet sample dataset which contains type 2 packets with a few isolated type 1 packets. The fact that most of the packets were classified as type 2 indicates that it contains long runs of a data type which was not expected to have long runs. Consequently it could not be compressed as well as it should have been with our current software. We could do somewhat better if we used a specific format for this data type instead of the generic type 2 format. Our software nonetheless achieved a respectable net compression factor of 4.72.

A typical run of 512 type 2 packets produced the following statistics:

Field #	Width	Meth 1 CF	Meth 2 CF	Meth 3 CF	Meth 4 CF	Meth 5 CF	Meth 6 CF	Meth 7 CF	Meth 8 CF	Best Meth	CF w/OH
1	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
2	1	512.00	256.00	32.00	28.44	5.57	5.57	0.86	0.87	1	73.14
3	4	n/a	n/a	0.75	2.29	2.14	2.12	1.25	1.18	4	2.27
4	9	512.00	256.00	144.00	135.53	42.67	42.67	7.55	7.65	1	307.20
5	10	512.00	256.00	150.59	142.22	46.55	46.55	8.37	8.48	1	320.00
6	16	512.00	256.00	178.09	170.67	67.15	67.15	13.13	13.30	1	372.36
7	16	512.00	256.00	178.09	170.67	67.15	67.15	13.13	13.30	1	372.36
8	6	n/a	1.49	1.19	1.43	1.36	1.34	1.33	1.32	2	1.49
9	8	n/a	n/a	1.05	3.07	1.86	1.67	1.88	1.72	4	3.07

Note that Field 9 contains 8 values per packet.

Table 6: Compression Results for Dataset from Tape AA

A 2841 packet data set, which contains type 3 packets. Our software achieved a net compression factor of 3.30.
A typical run of 512 type 3 packets produced the following statistics:

Field #	Width	Meth 1 CF	Meth 2 CF	Meth 3 CF	Meth 4 CF	Meth 5 CF	Meth 6 CF	Meth 7 CF	Meth 8 CF	Best Meth	CF w/OH
1	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
2	1	512.00	256.00	32.00	28.44	5.57	5.57	0.86	0.87	1	73.14
3	4	n/a	n/a	0.77	2.27	1.26	1.25	0.89	0.88	4	2.26
4	9	512.00	256.00	144.00	135.53	42.67	42.67	7.55	7.65	1	307.20
5	10	512.00	256.00	150.59	142.22	46.55	46.55	8.37	8.48	1	320.00
6	16	512.00	256.00	178.09	170.67	67.15	67.15	13.13	13.30	1	372.36
7	16	n/a	1.59	2.46	9.01	n/a	4.89	n/a	3.82	4	8.95
8	3	512.00	256.00	76.80	69.82	16.00	16.00	2.57	2.60	1	170.67
9	3	512.00	256.00	76.80	69.82	16.00	16.00	2.57	2.60	1	170.67
10	3	512.00	256.00	76.80	69.82	16.00	16.00	2.57	2.60	1	170.67
11	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
12	11	n/a	n/a	0.89	1.25	n/a	n/a	n/a	n/a	4	1.25
13	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
14	6	n/a	1.49	1.00	1.60	0.77	0.77	0.79	0.79	4	1.60
15	3	512.00	256.00	76.80	69.82	16.00	16.00	2.57	2.57	1	170.67
16	5	n/a	n/a	0.73	1.14	0.54	0.53	0.63	0.63	4	1.14
17	2	512.00	256.00	56.89	51.20	10.89	10.89	1.72	1.74	1	128.00
18	14	n/a	1.99	1.68	2.41	1.33	1.33	1.73	1.72	4	2.40
19	16	n/a	1.23	1.14	1.21	n/a	n/a			2	1.23

Table 7: Compression Results for Dataset from Tape BB

A 5000 packet encrypted data set. The data packets were mostly classified as type 1 It looks statistically random, which achieved a net compression factor of 0.999, indicating that the compressed data set was slightly larger than the original (because of overhead).

A typical run of 512 type 1 packets produced the following statistics:

Field #	Width	Meth 1 CF	Meth 2 CF	Meth 3 CF	Meth 4 CF	Meth 5 CF	Meth 6 CF	Meth 7 CF	Meth 8 CF	Best Meth	CF w/OH
1	8	n/a	n/a	0.80	0.98	n/a	n/a	n/a	n/a	none	1.00

Note that Field 1 contains 16 values per packet.

5. CONCLUSIONS AND RECOMMENDATIONS

Several algorithms such as Rice, LZ family and run-length codings were examined for selected digitized packet data streams. These algorithms are able to compress most data available with a fairly good compression factor except for tape A data and encrypted data. Tape A contains a mixture of several data formats in very short runs (several packets), rather than the long runs (typically hundreds of packets) for which the current version of our software was designed. For encrypted data, we are unable to compress any portion of the data, as expected. The remaining data sets were compressed with a compression factor of over 2.74 or more. This result is rather encouraging for this kind of data because most off-the-shelf software provides a compression factor of about 2.0. For tape D data, the LZ algorithms are superior to the others, since the data are highly repetitive and contain text strings. This data set also shows that LZ family of compression algorithms (dictionary search techniques) perform well even when the format is unexpected. As pointed out earlier, most of our software was developed for application to expected formats, but substantial compression must also occur with unexpected formats.

The next task is to reduce overhead bits, to improve predictions (to be used instead of prior values in first differencing) and to improve Rice coding, to try radix coding, arithmetic coding, another LZ78 algorithm, and a simple lossy compression algorithm, and, finally, to compress a meteorological database by developing a software variant for data in ACCESS or EMPRESS database format. Additional compression tasks will be implemented. The feasibility of real-time implementation on existing communication systems will also be examined with selected compression algorithms to prove that cost effectiveness and transmission efficiency can be simultaneously achieved.

6. REFERENCES

1. Lynch, T.J.: "Data Compression Techniques and Applications", A Lifetime Learning Publications, Belmont, California, 1985.

2. Held, G. and T.R. Marshall: "Data Compression Techniques and Applications: Hardware and Software Considerations", 3rd Edition, John Wiley & Sons Ltd., New York, 1991.

3. Sklar, B.: "Digital Communications: Fundamentals and Applications", Chapter 11, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

4. Berlekamp, E.: "Lossless Data Compression Algorithms: An Overview on Theory and Future Trends", Seminar Notes held at Roslyn, Arlington, VA., March 1989.

5. Nelson, M.: "The Data Compression Book", M&T Books, New York, 1992.

6. Global Grid Project Briefing Material (1992 Internal view graph), not officially published

7. Bailey, R.L. and R. Mukkamala: "Pipelining Data Compression Algorithms", The computer Journal, Vol. 33, No. 4, pp308-313, 1990.

8. Bell, T.C., J.G. Cleary, and I.H. Witten: "Text Compression", Prentice-Hall, 1990.

9. Tai, W.: "Data Compression-The End-To-End Information Systems Perspective for NASA Space Science Missions", Proc. of Space and Earth Science Data Compression Workshop, NASA Conference Publication 3130, Snowbird, Utah, April 1991.

10. Howard, P.G.: "The Design and Analysis of Efficient Lossless Data Compression Systems", Ph.D. Dissertation, Brown University, Technical Report # CS-93-28, June 1993,

11. Rice, R.F.: "Some Practical Universal Noiseless Coding Techniques", JPL Publication #79-22, NASA, JPL/CIT, Pasadena, CA March 1979.

12. Rice, R.F.: "Some Practical Universal Noiseless Coding Techniques, Part III, Module PSI14, K+", JPL Publication #91-3, NASA, JPL/CIT, Pasadena, CA, November 1991.

13. Ziv, J. and A. Lempel: "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. IT-23, No. 3, pp337-343, 1977.

14. Ziv, J. and A. Lempel: "Compression of Individual Sequences via Variable-Rate coding", IEEE Transactions on Information Theory, Vol. 24, No. 5, pp530-536, 1978.

15. Welch, T.A.: "A Technique for High Performance Data Compression", Computer, Vol. 17, No. 6, pp8-19, 1984.

16. Williams, R.N.: "Adaptive Data Compression", Kluwer Academic Publishers, Boston, MA, 1991.

17. Williams, R.N.: "An Extremely Fast Ziv-Lempel Data Compression Algorithm', Proc. of Data Compression Conference, 1991, pp362-371.

18. Rabbani, M. and P.W. Jones: "Digital Image Compression Techniques", SPIE Optical Engineering Press, Bellingham, WA, 1991.

19. "Specification for Processes and Associated Data for Microcats and PDMSS (PADMAP)", Naval Research Laboratory, Naval Center for Space Technology, #SSD-S-AS131C, April 1994.2