The Space Congress® Proceedings

1995 (32nd) People and Technology - The Case For Space

Apr 27th, 1:00 PM - 5:00 PM

Paper Session III-A - Data Compression and Error-Protection Coding

Robert F. Rice
Jet Propulsion Laboratory

DATA COMPRESSION AND ERROR-PROTECTION CODING

Robert F. Rice
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

ABSTRACT

"Data Compression" and "Error-protection coding" are two of the now widely heard but not well understood terms associated with the Information Super Highway. But this was not always so. Their familiarity is a consequence of developments which were initiated nearly 50 years ago with the introduction of modern information theory by Claude Shannon.

Both concapts and techniques which can dramatically improve the representation, storage and communication of digital data – the underlying component of modern information systems. Although often invisible to individual users, the commercial applications of compression and coding, which affect our daily lives now, have become extremely broad. Few of these applications can claim they were not directly or indirectly influenced by prior investments in this technology by NASA and the military.

This paper describes important specific on-going NASA direct technology transfers of data compression and error-protection coding techniques/technology. First jointly used to improve the return of Voyager images from Uranus and Neptune by a factor of 4, these techniques and their NASA sponsored custom high-speed microcircuits are now independently enjoying widespread use. A simplified laymen's description of these techniques and their performance characteristics is followed by a status on their technology transfer.

I. INTRODUCTION

You need not be Bill Gates to know that the computer revolution has had and will continue to have a dramatic impact on our daily lives. From hand-held calculators to virtual reality simulations in the "OJ" trial, the effects are pervasive.

As computer capabilities improve, as computers everywhere become linked to form the so-called Information Super Highway, more and more necessarily **digital** data of all types (but predominantly images) are being stored, processed and transferred. Consequently, the communication theory disciplines that make all this more efficient are also playing a much increased role.

The two disciplines addressed here are **data compression and error-protection coding.** Both are concepts which can dramatically improve the representation, storage and communication of digital data — the underlying component of modern information systems, embodied by the interconnection of computers and computer-like devices. Although often invisible to individual users, the commercial applications of compression and error-protection coding, which affect our daily lives have now become extremely broad. Few of these applications can claim they were not directly or indirectly influenced by prior investments in these technologies by NASA and the military.

This paper focuses on specific NASA developments in error-protection coding and data compression that are providing significant contributions to this computer/communication revolution. In the first case, NASA's application of a fundamental error-protection technique to the Voyager spacecraft acted as a catalyst for an explosion in both space and commercial applications. In the second case, specific adaptive data compression techniques that originated in the early 1970's, and were widely used within NASA have turned out to be quite fundamental themselves. Thus, 25 years later, refinements of these techniques and their NASA implementations are finding their way into the commercial arena too.

Subsequent sections will first develop some necessary technical background and then will describe the specific NASA efforts.

II. BACKGROUND

Standard Form Digital Data

Digital data of all sorts is represented in standard form as sequences of numbers. For example, a black and white square image can be thought of as a grid of numbers such that the higher the number, the lighter the area where the number is located. This is sort of like "fill-in-the-number" painting, except where colors are now **shades of grey** or brightness values. Each of the grid areas where the numbers are located are usually called picture elements or **pixels** for short. Figure 1 illustrates this idea.

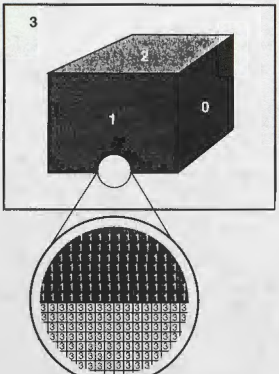


Fig. 1. Fill in the Shade

Brightness Resolution. In this crude example of a cube, there are only four possible shades of grey, as represented by the numbers 0 (black), 1, 2 and 3 (white). Clearly, having only four shades of grey to use limits the type of scenes that could be adequately represented. For example, gradual changes in brightness across such a cube or a face could not be distinguished. This limitation could be accommodated simply by increasing the possible shades of grey — but there is a penalty for this.

The integer numbers used to specify shades of grey are ultimately represented as binary sequences of zeroes and ones called **bits.** The collection of binary numbers used in this way is one form of a **source code.** A particular binary string is a **codeword.** The standard binary codes that are needed when there are a maximum of two, four or eight grey scale values to specify are illustrated in Tables 1 – 3, respectively.

Table 1. Standard Code for Two Greyscale Values

Grey Scale	Binary Codeword
0	0
1	1

Table 2. Standard Code for Four Greyscale Values

Grey Scale	Binary Codeword
0	0 0
1	0 1
2	1 0
3	1 1

Table 3. Standard Code for Eight Greyscale Values

Grey Scale	Binary Codeword
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Each table doubles the number of possible shades of grey that can be "resolved" but requires one extra bit (for each codeword) to do this. Thus, in general, the ability to handle 2^n shades of grey requires n bits per pixel (bits/pixel). Thus there is a consequence to simply increasing the number of shades of grey that can be handled. A digital system, such as your PC, will need to store and possibly transmit the extra bits.

Many shades of grey are not always necessary. When you "fax" a page from a letter, a scanner creates an image represented by only two shades of grey, black and white. This is acceptable since the vast majority of facsimile data contains black characters and lines over a white background. In-between grey scale levels aren't really necessary.

On the other hand, it is a generally accepted rule that 256 levels (8 bits/pixel) are needed to provide good quality black and white photographs and computer displays. Beyond that, the eye has difficulty perceiving any improvement. Color is handled by providing brightness values for each of three primary colors at each pixel location. Your eye averages brightness values to create intermediate colors. So called true color capability is provided when a display system can support 8 bits of color brightness for each of the primary colors.

The imaging instruments for Voyager and Galileo were both developed with 8 bits/pixel grey scale capability. The original Mariner 4 used six. But note that in many science applications digital data processing techniques can often be used to enhance displays so that very tiny changes can be viewed **and measured.** Sometimes science data is never viewed in its raw form at all. The Galileo imaging-like instrument called NIMS uses 10 bits to specify its grey scales and the Cassini imager uses 12 bits/pixel. At the other far end of the application spectrum, certain star-field images are maintained as 32 bit numbers.

Spatial Resolution. Twenty bits of grey scale or color resolution won't do you any good if you have only one pixel to describe an object.

Figure 2 illustrates what happens when the number of pixels used to describe a black right triangle is quadrupled (doubled in both dimensions). In the first case a grid of 8x8 pixels describes the triangle. The description is quite poor, making the sloping line look more like a staircase. This is said to be a case of the "jaggies". The adjacent grid has been increased to 16x16. Although the jaggies have not disappeared, the sloping line looks a lot better.

But again, there is a penalty for this increased quality. Each additional pixel used to describe a scene means additional data bits are required. In this example, the number of bits used to describe the triangle scene was increased by a factor of 4. What happens if the triangle is only a small fraction of a complete image?

Image Parameters. The parameters of a digital image can be kept in mind by the rectangle in Fig. 3. It shows an image made up of L lines of P pixels, each represented by n bits for grey scale and/or color. The total number of bits for an image is then

$$B = n\,PL \text{ bits} \qquad (1)$$

Or, using the more familiar computer term "byte" (where 1 byte = 8 bits)

$$B' = \frac{n\,PL}{8} \text{ bytes} \qquad (2)$$

Table 4 provides examples of these digital parameters for various applications. Total bits are shown in millions, Mbits ("Megabits").

Temporal Resolution. But it gets worse. Movies and television scenes that you are familiar with are the result of many still images (frames) that are placed before your eyes in rapid succession. The American television standard for frame rate is 30 frames/sec, fast enough so you don't see **flicker.**

In most cases, 30 to 60 frames/sec can be expected to set requirements for digital systems too. (In special science applications, thousands of frames/sec are considered.) Thus the majority of commercial applications will be burdened with the rapid accumulation of a lot of bits. The raw digital data rate which generates the image frames in Fig. 3 at f frames/sec is, from Eq. 1

$$D = f\,B = f\,n\,PL \text{ bits/sec} \qquad (3)$$

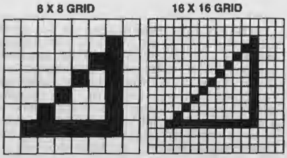


Fig. 2. Spatial Resolution

In the extraordinary case of a future (high-end) digital movie camera, this raw data rate reaches 8,640 Mbits/sec.

Observation. In the future, almost every data source can be expected to be represented in digital form as depicted above. Certainly we are seeing the results of a constant competitive thrust to improve quality and initiate new applications not feasible before. Techniques and technology that can reduce the burden of this ever increasing explosion in digital data can only become more important.

Table 4. Image Parameters

Application	L (lines)	P (pixels)	n (bits/pixel)	B (M bits)
Facsimile @ 200 pixels/in.	2200	1700	1	3.74
Facsimile @ 1000 pixels/in.	11,000	8,500	1	93.5
CGA graphics	200	320	2	0.128
(Super) VGA graphics	480	640	8	2.45
HD TV	1200	1024	24	29.5
Digital Movie	3000	4000	24	288

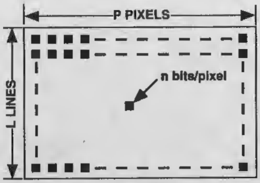


Fig. 3. Standard Image Parameters.

Data Compression

A very simple illustration of the basic ideas of data compression is provided in Fig. 4 and Fig. 5.

In Fig. 4 a compression "black box" compresses a data set, X (represented in some standard form as described above) into another "compressed" sequence of bits, Y. Usually Y will use considerably fewer bits than X. The reduction factor is called the Compression Factor, CF. So if X requires N bits, Y requires N/CF bits.

Potential Benefits. The primary beneficiaries of a CF:1 reduction in the number of bits are the system components which store and communicate data. These are shown inside the dashed box in Fig. 4. Without worrying about some practical details.

- A fixed amount of memory could store as much as CF times as many data sets; or memory requirements could be reduced by as much as CF:1.
- A fixed communication link could transmit data sets CF times as fast; or transmission requirements could be reduced by as much as CF:1 without slowing the transfer of data.

Types of Compression. In most cases, compressed data must be "decompressed" or "reconstructed" back into a **version,** X', of the original data before it can be used. When the reconstructed version is identical to the original, the compression process is called **lossless** (also called lossless coding). Otherwise, of course, the process is called **lossy.** These points are illustrated in Fig. 5 which plots reconstructed **Data Quality** vs. the number of bits used to compress.

Point A corresponds to "no compression". Quality is viewed as perfect (same as original) and N bits are required.

Point B corresponds to the application of a lossless compression technique. Quality is still perfect but the number of bits has been reduced. Here the reduction shown is about 2:1 which is **typical** for non-binary grey scale images. But in general, the lossless compression factor will vary and achievable compression factors can be dramatically different from one application to the next. More on this later.

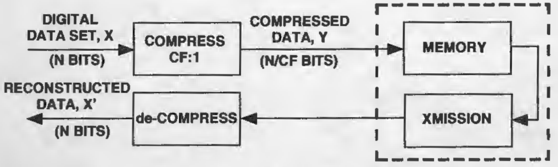


Fig. 4. Compression/Decompression

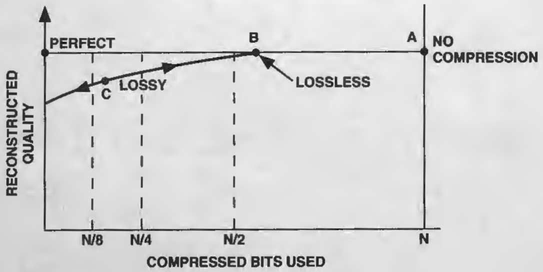


Fig. 5. Quality vs. Bits Used

Point C in Fig. 5 is much harder to understand. Basically, to get better compression factors than provided by a good lossless compressor at point B, some changes in the reconstructed data must be accepted. The larger those changes are allowed to be (lower quality), the larger the compression factor that should be achievable. The tradeoff is indicated by the arrows on the graph near point C.

The goal in designing lossy compressors is to achieve the highest compression factor which results in reconstructed quality that satisfies the requirements for a particular application. Unfortunately, quality is a difficult term to define once the data is no longer perfect. In practice it is usually a combination of subjective measures as well as quantitative ones. This leaves a great deal of room for interpretation, hand waving and competition for the best approaches.

Even with agreement on how to specify quality, the requirements for how much of that quality is needed is data dependent, application dependent and user dependent. For example, preserving every little detail in a lung x-ray could be quite important. But would you need a perfect insignia on Arnold Schwarzenegger's rifle butt, coming at you at 30 frames/sec?

Regardless of these issues and other practical difficulties such as implementation, many advanced lossy techniques employed today demonstrate real and significant advantages. Multi-media images are often communicated and viewed at compression factors ranging from 4:1 to 100:1. But there are more advances to come.

Error Sensitivity. A "bit error" in the communication or storage of digital data means that a '1' has been turned into a '0' or vice versa. Bit errors can occur individually or in bursts. The impact of such errors depends a lot on the type of data. An error in an image pixel's standard representation will affect only that pixel. By contrast, the impact of an error on compressed data can be far more damaging. Typically, a decompressor will become confused, causing sometimes significant errors in reconstructed data until the decompressor is somehow "reset".

This fact hindered the practical application of data compression for years, both in space and in commercial applications. Simple solutions often resulted in a loss of communication efficiency which eliminated much, if not all, of the benefits provided by data compression in the first place. Advances in error-protection coding techniques and technology have played a significant role in solving this problem.

III. LOSSLESS DATA COMPRESSION

Remember when "faxing" a page took several minutes? While there has been dramatic improvements in modem data rates, modem facsimile almost owes its success to lossless compression. Compression factors of 20:1 can be expected on typical facsimile documents using standardized compression techniques.

In a related application, Data Tree Corporation of San Diego, California has created perhaps the largest and fastest growing document image data base in the world while seeking to automate the Title (and other) Industry. Customers throughout California can almost instantly access documents that once took days to acquire. None of this would be feasible without the application of lossless (facsimile-like) data compression.[1]

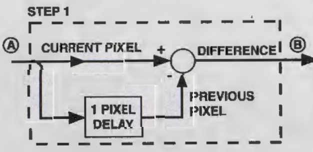
Grey Scale Images

Unfortunately, achieving high compression factors associated with facsimile lossless compression is an exception, rather than the rule. When there are many shades of grey to represent (meny bits/pixel), the rule of thumb for anticipated compression factors is more like 2:1. But this is still significant considering the gazillions of bits being generated today.

Bell Labs dominated the compression development efforts going on in the late 1960s and early 1970s. The primary focus at that time was in lossy compression of images for early picture phone applications. But these efforts did establish a basic predictive approach to the lossless compression of images and image-like data as shown in Fig. 6.

This is easier to understand than it looks. In the first step, differences between adjacent pixels are computed. Smaller differences will tend to occur more frequently than the larger differences because image data tends to stay the same more than it tends to differ. More generally, this is really a special case of a **predictor** in which the current pixel is predicted to be the same as the last. The differences can be viewed as the errors in these predictions.

In the second step, we wouldn't gain anything by using a standard **fixed length code** to represent the differences as in Tables 1–3. Instead a **variable length code** is applied so that shorter codewords are used for smaller differences. Since the smaller differences occur more frequently, the shorter codewords are used more frequently. Then, **on the average**, the number of bits used is reduced. Hence, compression is achieved. (The second column in this table numbers the differences, starting at zero. We will return to this later.)



STEP 2: VARIABLE LENGTH CODING OF DIFFERENCES

DIFF	DIFF NO.	CODEWORD
0	0	00
+1	1	01
-1	2	100
+2	3	101
-2	4	11100
+3	5	11101
–	–	–
–	–	–
–	–	–
–	–	–
–	–	–

CODED DIFFERENCE

Fig. 6. Basic Prediction and Variable Length Coding

D.A. Huffman long ago provided a now famous algorithm for creating an "optimum" variable length code for a known distribution (i.e., a known frequency of occurrence of differences or other data).[2] For many, this meant there were no more problems to solve. Just generate a **Huffman code** and you were assured of "optimum" performance. If it wasn't a Huffman code, it wasn't worth considering.

But in the non-academic world, real data hardly ever produces static data characteristics. A Huffman designed code could in fact perform poorly (relative to other possible codes and information theoretic performance measures) when the actual distributions differed significantly from the design distribution. This was often the case.

Further implementation requirements for today's data can rapidly get out of hand. Suppose input at point A in Fig. 6 was represented with a typical 8 bits/pixel (256 shades of grey). The number of possible differences at point B is 511. Then the lookup table for step 2 in Fig. 6 must maintain a codebook of 511 codewords. And applications with more than 4096 grey scale values is not uncommon today. As a real example, the basic structure in Fig. 6 was implemented for the ill-fated Mars Observer spacecraft. Roughly 20,000 bits of memory was reserved to store the codebook.

A technique that eliminated these and other shortcomings was proposed back in 1970.[3] Hardware prototypes were designed both by JPL and TRW. The TRW design was capable of 20 Mbits/sec, very fast at the time.

Over the following two decades, variations and extensions to this work were developed and quietly applied directly to other imaging applications, to other instrument data and as part of lossy image compression algorithms. The latter results were tied into a patented joint data compression/error-protection concept in the mid-1970s called the Advanced Imaging Communication System (AICS). AICS is discussed in a later section.

Isolating the (Source) Coding Problem. The center column of non-negative integers in Step 2 of Fig. 6, appear to simply number the differences. But they serve a more important purpose. Note that because a difference value can always be calculated from an integer, we might just as well code the integers. Because of the preceding steps of prediction and differencing, sequences of integers look like a data source from which the smaller integers occur more frequently than the larger. **Any** variable length code that might be applied should use its shortest codewords for the smallest integers. Additionally, the prediction **makes the integers independent of each other** (knowing one integer doesn't help you guess at the value of another). We are free to focus on the coding problems associated with the integers. In doing so there is no real need to know that the integers came from image data or that prediction and differencing were involved.

A non-negative integer data source that behaves as this one (ordered frequency of occurrence and independence) is called a **Standard Source**. The steps that produced it here are called preprocessing. But this Standard Source can be created from a great many seemingly distinct applications by **proper preprocessing**, which may take many forms. Hence, **the solutions to coding problems of the Standard Source should enjoy wide applicability.**

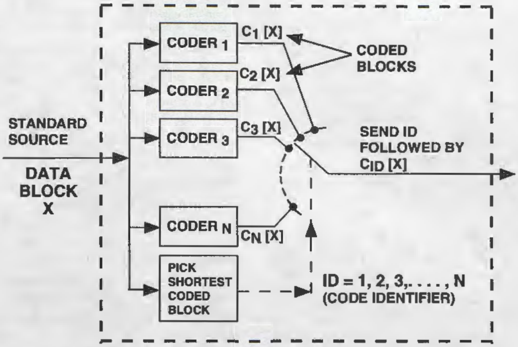
Coding the Standard Source.

Information Theory provides us with a mathematical measure of source coding efficiency called **entropy**. Basically, when applied here, a Standard Source with a **fixed distribution** (frequency of occurrence) of the integers cannot be coded with fewer bits/integer than the entropy (and still be lossless). A coder which codes **close to the entropy**, such as a Huffman code used on its design distribution, is said to be **efficient**. For image applications, activity/detail go hand-in-hand. Higher detail implies higher entropy. Real problems typically produce wide ranges of entropy so that a Huffman code may be efficient only part of the time.

The range of efficient performance can be extended by the concept of an "Adaptive" Variable Length Coder (AVLC) as shown in Fig. 7. This structure is a fundamental part of the so-called **Rice Algorithms**. Here, short blocks (e.g., 8 to 16 integers) provide the input. A decision mechanism determines which of several variable length coders to use by determining the coder that will produce the shortest compressed block. A short identifier code (e.g., 2 to 4 bits) precedes the selected compressed block to tell a "decoder" which code had been used.

Implementing an AVLC with ten or twelve Huffman codes, requiring over 4000 codewords each, doesn't sound like a very practical undertaking. It's not! Here's how its done.

We start with the simplest to implement variable length code. A codeword for integer N is simply N zeroes, followed by a '1'. For example, the codeword for number 8 is '000000001'. The codeword for zero is simply '1' (no zeroes, followed by a '1'). Clearly, the ability to easily generate each codeword means that a table to store this code is unnecessary. This **"Fundamental Sequence"** code is efficient over the entropy range from 1.5 to 2.5 bits/integer.



STANDARD SOURCE
DATA BLOCK X
CODER 1
CODER 2
CODER 3
CODER N
C1 [X]
C2 [X]
C3 [X]
CN [X]
CODED BLOCKS
SEND ID FOLLOWED BY CID [X]
PICK SHORTEST CODED BLOCK
ID = 1, 2, 3,. . . . , N (CODE IDENTIFIER)

Fig. 7. Adaptive Variable Length Coder Concept

Other code options, called **Split-sample Options**, are equally simple. Suppose that n bits are required to represent input integers by standard fixed length binary codes as in Tables 1-3. Assume bits are numbered from right to left so that the rightmost bit is bit #1 and the leftmost bit is bit #n as in

$$\underbrace{\boxed{\begin{matrix}\text{BIT}\\ n\end{matrix}}\boxed{\begin{matrix}\text{BIT}\\ n-1\end{matrix}}\;\_\;\_\;}_{\text{MOST SIGNIFICANT } n\text{-}k}\underbrace{\boxed{\begin{matrix}\text{BIT}\\ k\end{matrix}}\;\_\;\_\;\boxed{\begin{matrix}\text{BIT}\\ 2\end{matrix}}\boxed{\begin{matrix}\text{BIT}\\ 1\end{matrix}}}_{\text{LEAST SIGNIFICANT } k} \qquad (4)$$

Now suppose that an integer's binary representation is split into two parts. The first k bits are called **Least Significant** bits (LSB) and the remaining n-k bits are called **Most Significant** bits (MSB).

For every value of k from 0 to n, a code option is created by performing the following operations on **each** integer of a data block:

- Split off the n-k Most Significant bits and use the Fundemental Sequence code on them (treated as an integer) to generate a codeword (5)
- Tack on the k Least Significant bits (unchanged) onto the end of the codeword generated in (5) (6)

That's it. Again no tables are required. Further, decisions over which code option to use can be based on how many bits the Fundamental Sequence code will use by itself (k=0). . . . which can be determined easily without actually coding the data. So how do these Split-sample Options perform?

These and other slightly more complex Split-sample Options have been coding close to the entropy within the adaptive structure of Fig. 7 for over two decades. But Pen-shu Yeh of Goddard Space Flight Center (GSFC) tied down the performance characteristics for a specific, well accepted model for how image data occurs.[4] She showed that the Split-sample Option with k split LSBs has a range of efficient (close to the entropy) performance of about 1 bit/integer and centered at an entropy of

$$k + 2 \text{ bits/integer} \qquad (7)$$

Thus the AVLC can include a coder option which performs efficiently at any entropy above 1.5 bits/integer. (Other options not discussed here address the lower entropies.)

More amazingly, Yeh showed that these easily generated **Split-sample Code Options are equivalent to Huffman codes.**

<u>Performance Comparisons</u>. Today's best alternatives to the Rice Algorithms for image-like data include LZW, Adaptive Huffman and Arithmetic Coding. LZW stands for Lempel, Ziv and Welch, the three inventors. It is most effective on data sets for which very little is known ahead of time. Most

computer disk compression programs are based on LZW. Adaptive Huffman is an adaptive algorithm where the code used is constantly updated. Arithmetic is a sophisticated approach often associated with International Business Machines (IBM).

Each of these is compared to the Rice Algorithms for a broad range of image data in Fig. 8 by Jack Venbrux.[5] In these comparisons, each compression technique used the same Rice Algorithm preprocessing. The results speak for themselves. Later papers by Venbrux and Yeh also show a significant advantage over the lossless mode of **JPEG**, the commercial standard for still image compression.

Implementations. Lengthy computer-driven slideshows were used to describe all aspects of these adaptive coding techniques at a NASA sponsored data compression conference in 1988. A committee strongly recommended that NASA should build such a coder/compressor in high-speed custom microcircuits for broad general use.

At the time, Warner Miller of GSFC was already directing efforts to develop other specialized microcircuits at the NASA Micro Electronics Research Center (MRC), then located in Moscow, Idaho. To assure broad utility within NASA and possible commercial applications he recognized the need to preserve the flexibility inherent in the algorithms and to provide a compatible **decoder/decompressor**.

The initial coder chips were designed by Jack Venbrux of MRC and tested in 1991. They included a predictive preprocessor like in Fig. 6, but with the flexibility to externally replace the built-in predictor and to enter data directly into the AVLC if desired. The AVLC incorporated 12 code options and could operate with 4 to 14 bit data. Some of the chips operated successfully in the lab at **900 Mbits/second**. Coder **and** decoder chips of this initial design are now sold commercially by Advanced Hardware Architectures, Inc., now located in Pullman, Washington. Second generation chips with additional capability were recently made available.

During the same time period, the Jet Propulsion Laboratory (JPL) also developed a first generation "coder" chip and tested it in 1990. Driven not by long term goals but by project constraints, numerous "unnecessary" features/flexibility were eliminated, drastically limiting its general use. A second generation flight chip has restored some of these features. But a decompressor is still "someone else's problem".

Incredibly, while both of these powerful chip developments were proceeding, one work-group was busy inventing predictive, non-adaptive lossless compressors, starting with the diagram in Fig. 6.

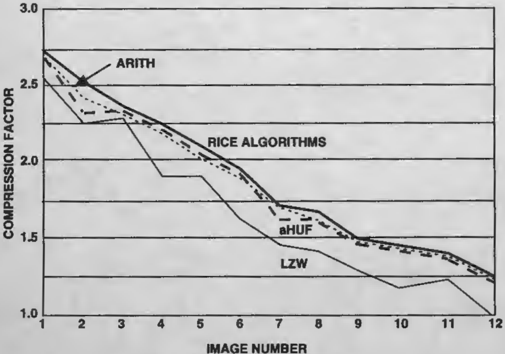


Fig. 8. Performance Comparisons

Observation. The list of hardware and software applications of the Rice Algorithms is rapidly growing. The potential for widespread commercial use is now significant. Much of this attention and proof of concept must be credited to the farsighted technical management of Warner Miller and his competent team, particularly Pen-shu Yeh and Jack Venbrux.

IV. ERROR PROTECTION CODING

As of the mid 1970's error-protection coding was already an important part of the Deep Space Network (DSN) to communicate efficiently with deep-space probes. The baseline coding system at the time was a so called short-constraint length convolutional code with Viterbi decoding. (The coding/decoding terms here now refer to error-protection.) Viterbi decoding is a powerful decoding technique named after Andrew Viterbi, one of the most famous contributors to communication theory and practice. Systems like this have enjoyed extremely widespread use.

Back then, image data was uncompressed and therefore fairly tolerant of communication errors (one error affecting one pixel). The design criteria was to have no more than 1 error in 200 bits. Compared to using a communication system without coding, the convolutional coding system could achieve this desired error rate at roughly twice the transmission rate.

Non-imaging data tended to be error-sensitive and hence required a lower error rate than uncompressed imaging did. This could be accomplished by simply lowering the transmission rate being used with the convolutional system (errors decrease because the signal-to-noise ratio increases with decreasing data rate). But this would mean that the image data, **which usually constituted more than 90% of all data** would have to be transmitted at a lower transmission rate too. The fix was to apply another kind of error-protection coder called an algebraic coder to the non-imaging data. The Golay coder used, worked on blocks of 24 bits. For every 12 bits of real data, another 12 "parity" bits were added. (The parity was used to correct some of the bit errors that might occur in the 24 bits.) Thus the effective data rate through this coder was cut in half. But it accomplished its purpose, the added parity reduced the error rate (on the 10% or less, non-imaging data) from 1/200 to about 1/100,000. The 50% overhead did not matter, since it was only applied to 10% of the data.

From an error-protection point of view, this was the essence of the baseline communication system for Voyager, Galileo and many other study projects.

Trying to suggest the use of image data compression with this system or an uncoded one was not well received (to say the least). For example, suppose the Rice Algorithm lossless compression technique were applied, yielding an average 2:1 reduction in the number of data bits that needed to be transmitted. Because of the error-sensitivity mentioned earlier, the communication error rate needed to be reduced. But this could only be accomplished by reducing the transmission rate by (about) a factor of 2:1, like for the non-imaging data. Overall there would be no gain. Another solution was needed.

Back in the early 1960s, Irving Reed and Gus Solomon formulated a specific class of algebraic code, now known as Reed-Solomon codes. They were primarily an interesting academic topic for many years. But in the late 1960s, key work in decoding algorithms, notably by Elwyn Berlekamp, paved the way for practical applications. Following this, J.P. Odenwalder, with Viterbi as his advisor, wrote his PHD thesis on combining (called concatenation) very large Reed-Solomon codes (e.g. 2000 bits long) with the convolutional codes. This was followed by additional studies by Linkabit Corporation and Ames Research Center in 1972.

One particular combination of Reed-Solomon code concatenated with the Voyager/Galileo convolutional code would produce "virtually error-free" communication (one error every billion bits) at almost the same transmission rate as the convolutional channel alone would produce errors at 1/200. Linkabit work suggested implementation was feasible too.

But why implement it? Imaging scientists "wanted an error rate of 1/200", didn't they? A powerful communication system that did this was already in place. Besides, there was another powerful alternative for achieving low error rates already being used on Pioneer spacecraft too, in the form of Sequential Decoding. These issuea and a few hundred "red herrings" stood in the way.

Then in 1973 an **Advanced Imaging Communication System (AICS)** was proposed.[6]-[8] It combined the possibility of data compression on all data with the concatenated Reed-Solomon/conv.-Viterbi data link. The Golay could be discarded. The focus of the AICS proposal was in demonstrating true end-to-end advantages by this combination to the real customer, the scientist. In conjunction with another Linkabit study, any legitimate arguments against the concatenated link were dispelled — to most. Basically, the concatenated link eliminated the data rate penalty usually associated with data compression, and solved a few other problems along the way.

The adaptive, rate controlled, lossy (and lossless) image data compression originally used to demonstrate AICS potential incorporated many new innovative features, some of which have not yet been adequately addressed. It suffices to note here that science value studies in 1975 and 1980 firmly concluded that real end-to-end advantages of AICS equipped with sophisticated data compression could range from 4 to 10:1. A patent was issued for AICS in 1976.[9]

In 1977 a herdware Reed-Solomon coder was placed on Voyager just prior to launch. Later at the Uranus and Neptune encounters, a software, simplified Rice Algorithm was used to represent Image data. The combination of Reed-Solomon coding end the elimination of the Golay parity bits, provided Voyager with a nearly 4:1 improvement in the ability to transmit perfect images.

Since then the concatenated Reed-Solomon/Conv.-Viterbi data link achieved NASA and international standerdization, with AICS primery code parameters intact. For additional historical informetion on this subject consult the congressional study in Ref. 10.

Once "virtually error-free" communication was a reality, packet telemetry considerations beceme practical. This led to further standardization, primarily through the persistent efforts of Adrian Hooke and Edward Greenberg.

Early exposure of AICS implementations initiated an explosion of activity in Reed-Solomon code applications (with and without convolutional codes).[11] There are now two distinct code structures: the standard architecture and the Berlekamp Architecture (which drastically reduces the size of discrete part implementations).[12] Marv Perlman provided a patented technique for converting between them.

Again, Wamer Miller of GSFC, long aware of the general importance of Reed-Solomon codes, directed the development of high-speed Reed-Solomon microcircuits at MRC, now located in Albuquerque, New Mexico. Technology transfer of this work to industry now includes names like Hewlett Packard, IBM, TRW, Lockheed and Loral.

ACKNOWLEDGMENTS

The author would like to offer thanks to key individuals and organizations whose contributions and support have helped achieve the results described herein. To Alan Schlutsmeyer for his extraordinary programming talents. To Warner Miller, Pen-shu Yeh, Jack Venbrux, Ed Hilbert and Jun-ji Lee for their technical and moral support. To Gamo Publications, Burbank, California for their always professional publication support. When no one else could get it right — they did.

Some of the research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration.

REFERENCES

1) Paul Johnson, Private Communication, Data Tree Corporation, San Diego, California.

2) D.A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," **Proc. IRE**, Vol. 40, pp. 1098-1101, 1952.

3) R.F. Rice and J.R. Plaunt, "Adaptive Variable Length Coding for Efficient Compression of Spacecraft Television Data", **IEEE Trans. on Communication Technology**, Vol. COM-19, Part I, Dec. 1971, pp. 889-897.

4) Pen-shu Yeh et. al., "On the Optimality of a Universal Lossless Coder," **Proceedings of the AIAA Computing in Aerospace 9 Conference**, San Diego, California, October 19-21, 1993.

5) J. Venbrux and Pen-Shu Yeh, "A VLSI Chip Set for High-Speed Lossless Data Compression," **IEEE Transactions on Circuits and Systems for Video Technology**, Vol. 2, No. 4, Dec. 1992.

6) R.F. Rice, "Channel coding and data compression system considerations for efficient communication of planetary imaging data," Chapter 4, **Technical Memorandum 33-695**. Jet Propulsion Laboratory, Pasadena, CA, June 15, 1974.

7) R.F. Rice, "An advanced imaging communication system for planetary exploration," Vol. 66 SPIE **Seminar Proceedings**, Aug. 21-22, 1975, pp. 70-89.

8) Robert Rice, "End-to-End Information Rate Advantages of Various Alternative Communication Systems," **JPL Publication 82-61**. Jet Propulsion Laboratory, Pasadena, California, September 1, 1982.

9) AICS Patent 3988677, Oct. 26, 1976.

10) -------, "The Economic Impact and Technological Progress of NASA Research an Development Expenditures," prepared for the Academy of Public Administration, Washington, DC by Midwest Research Institute, Kansas City, Missouri, Sept. 20, 1988.

11) S. Wicker, et. al., "Reed-Solomon Codes and Their Applications," **IEEE Press**, 1994.

12) R. Wilson, "Error Correction for the Masses," Electronic Engineering Times, Nov. 21, 1994.