

Lossless Image Compression – General Overview cmpe 263 Winter 2001

Glen G. Langdon, Jr.
University of California, Santa Cruz

For Monday, February 12, 2001

Some Theoretical Preliminaries related to probability distributions and coding

Lossless image compression depends entirely on statistical techniques, i.e., compression depends upon coding each outcome i of a statistical event E according to its own relative frequency $p(i)$. The self-information $l(i)$, measured in bits, of an outcome i is $\log(1/p(i))$ (also written $-\log p(i)$) where $p(i)$ is the relative frequency. Thus, the coding part of the compression problem must, for 100% coding efficiency, increase the code-length by $l(i)$ bits for each outcome i . The point to be made here, is that the encoding part can only worsen the achievable compression of a particular approach.

The achievable compression is increased when the number of probability distributions is increased. If we take a file of data D of N 8-bit symbols and determine the symbol frequencies $n(i)$ for the single probability distribution, we determine the self-information $l(i)$ for each symbol (i) as $\log_2\{(N/n(i))\}$. Multiplying the self-information of a symbol by the number of times that symbol occurs gives that symbol's contribution to the code length. Summing the contribution of each symbols gives the ideal code length $I(D)$. Viewed another way, the ideal code length of a given distribution of N symbols is:

$$I(D) = [\text{sum over } i, 0 \text{ to } 255] n(i) \times l(i).$$

In lossless compression, the single most important factor to success is the shape of the histogram that characterize the several probability distributions of the statistical events. In dealing with large amounts of data in a single file, the more the algorithm is arranged to use more than a single distribution, the better the compression. The context model partitions the data into independent distributions.

Suppose we use a local context model to split the N data items into two "piles", one for context 0 and the other for context 1. For example, let one

pile, context 0, be for symbols that follow a value from 0 to 127. The other pile, context 1, is for symbols that follow a value from 128 to 255. For images, we would expect the average value in context 0 to be less than the average value of the entire symbol set, since the intensity value from one symbol to the next does not change very much. For a given image let $N(0)$ be the number of symbols falling in context 0, i.e., $N(0)$ is the sum of values $n(i|0)$ where $n(i|0)$ denotes the number of times symbol "i" in context 0. The self-information $l(i|0)$ is $\log(N(0)/n(i|0))$.

The ideal length for context 0, denoted $I(D|0)$, is:

$$I(D|0) = [\text{sum over } i] n(i|0) \times l(i|0).$$

$I(D|1)$ is similarly determined. After calculating $I(D|0)$ and $I(D|1)$, it is not possible for $I(D) < I(D|0) + I(D|1)$. In other words, from the mathematics of the ideal code length, you cannot lose. However, in practice, the overhead (which does not appear in the ideal length equations, can cause a loss in compression when context populations are sparse, i.e., when a context C is visited infrequently and $N(C)$ is a small number.

The overhead is the cause of the "sparse context problem". In practice, the decoder must use the same coding probabilities as the encoder used. The explicit transmission (eg. in the header) of coding parameters costs additional bits. These additional bits are often add more to the code string than is saved by the use of the sparse context. In the one-pass adaptive approach, the overhead is caused by "learning" the distribution.

In splitting distributions, the mathematical calculation can "break even"; i.e., it is possible that for a given data file D

$$I(D) = I(D|0) + I(D|1)$$

The case of equality occurs only if splitting the distribution via contexts 0 and 1 yields identical distributions to the single distribution comprising all of D. In other words, no compression gain is achieved by contexts if the symbol occurrences are independent of the chosen contexts. This result is the definition of statistical independence; if $p(A) = p(A|B)$ then the probability of A is independent of the value of condition (context) B.

A third aspect of compression deals with the question "how does the decoder determine the context-dependent coding probabilities"?

The answer deals with how to characterize (or model) the distribution and then how to estimate the parameters or probabilities. In most cases the algorithm itself embodies the type of distribution as a histogram, and the parameters

become the probability values for each symbol. There are three general strategies for how the decoder determines these probabilities: (1) static, (2) two-pass, and (3) one-pass.

The static approach provides for a fixed code that embodies a static distribution, or a selection of one of a number of static codes. The static codes employ "average" or "typical" distributions built into both the encoder side and decoder side.

A two-pass technique transmits the code or distribution parameters along with the codestring, after the encoder makes a first pass over the data to determine the contexts and/or distributions.

A one-pass probability estimation algorithm requires the encoder to use an adaptive (dynamic) technique that counts symbols instances from the "up-to-now" encoded data, and employs this information "on the fly", while the decoder does the same "on the fly" algorithm acting on the "up-to-now" decoded image.

Cleary and Witten [] argue that there is not much compression advantage or difference between the one-pass and two-pass techniques. Their argument shows that the portion of the final codestring assignable to the overhead of transmitting the statistics to the decoder is approximately the same as the portion of the final codestring assignable to learning the statistics "on the fly".

Various one-pass counting strategies have been published; one is called the Laplacian estimator after Laplace's "Rule of succession": simply initialize each count to 1, then add 1 to each symbol's count after encoding (decoding) that symbol. The resulting frequencies model the distribution as a histogram.

Codes for one-pass approach: arithmetic or adaptive Huffman

The term "adapter-coder" suggests an algorithm for the coding part that incorporates the probability estimator function as well, to provide a single unit at the encoder side and the decoder side that have an "adapter algorithm" to update or learn the coding parameters needed by "coder algorithm".

When the coder employs a Huffman code, the adapter-coder is called an adaptive Huffman code. Faller [fal73] was the first to design an adaptive (one-pass) Huffman code, and Gallager [gal78] publicised the code. Many, including Knuth and Vitter made subsequent suggestions. Adaptive Huffman codes dynamically maintain the binary "tree" employed in the design of static Huffman codes. As certain branches become more popular than others, the subtrees may be exchange and modified. A key idea to maintaining the tree's data structure is called the "sibling property" by Gallager. The adaptive Huffman code of Faller

preceded adaptive arithmetic codes.

Arithmetic codes use, or can use, the probabilities directly in the coding process, or with some multiplication and division, use the counts value themselves directly in the coding process. There are many ways to incorporate the count-ratio and scaled-count techniques into the arithmetic code. Perhaps the first such is described in [lr81], where adapter SSS-adap provides the coding parameter needed by the "shift-coder" also described in [lr81].

Several

techniques are reported in the IBM Technical Disclosure Bulletin. For many

(m-symbol) alphabets, and arithmetic adapter-coder is described by Goertzel in "File Compressor" [goe87].

The standard distributions employed by statisticians and others, such as the Poisson, Gaussian, or Laplacian distributions, have shapes that can be economically communicated to the decoder via a small set of parameter values. For example a Gaussian distribution is completely characterized by specifying two parameters: the mean and the variance. For the static approach, placement of these parameter(s) in the header of the code string permits the decoder to design the code (given that the decoder knows the proper algorithm).

Overhead and too many coding parameters

A key issue related to the overhead required for the decoder to use the same context-dependent coding parameters (symbol probabilities, codewords, index to

a probability, parameter values of a standard parameterized distribution) is the following: the overhead is related to the total number of coding parameters provided by the context model as well as how the context-dependent distributions are modeled. This concept is explicit in the title of [lrt80]. The concept of managing parameters cuts across all data compression applications using higher-order contexts, not just the image compression application.

In [rl81], it was shown that the proper way to compare two approaches in a fair way is that both approaches employ the same number of coding parameters. Otherwise the approach bringing more distributions to bear on the compression problem should win out. The problem arose in how to fairly compare an n-gram approach to the symbolwise approach to data compression. The n-gram or "blocking" or "alphabet-extension" technique is a technique that increases the alphabet size by applying a single probability distribution to the n-symbol blocks. For example, given a block size of 2, the single "extended" alphabet now consists of all possible symbol pairs. Shannon's term for a 2-symbol block is a bigram, and the term for an n-symbol blocking or alphabet extension is the n-gram.

For compressing bigrams, in effect the context model is seen to be the following. The first symbol of the block is compressed under a memoryless source model, and the second symbol is compressed under the context that it follows the first order symbol. In [rl81], we showed the symbolwise equivalent of blocking uses the same number of coding parameters.

Coding issues for binary code alphabets

The binary alphabet prevails today as the code alphabet for compression on computer systems.

Huffman codes

Huffman coding is an integer-length code, i.e., the codeword for each outcome has an integer length in bits. The "coding probability" of a Huffman code of length L is 2^{-L} . For example, a 4-bit codeword represents 1/16 of all binary strings (those binary strings that begin with the first four bits of the codeword). Hence a 4-bit codeword uses up 1/16 of the remaining available code space at the point the codeword is used.

Coding efficiency for prefix (Huffman) codes

If a probability distribution, by lucky chance, only has probabilities of the form 2^{-L} (L a positive integer) then the Huffman codewords are 100% efficient for that distribution. The self-information for an outcome of probability 2^{-L} is exactly L bits. When L is not an integer then the self-information length for that outcome contains a fractional part.

If a symbol's self-information is 1.5 bits, then its probability falls between 1/2 and 1/4, but the symbol's prefix codeword length can only be either 1 or 2. Encoding outcomes of self-information 1.5 with either 1 bit or 2 bits is very inefficient. Less of a problem, but still inefficient, is a self-information value of 2.5, which calls for a prefix codeword of either 2 or 3 bits in length.

At symbol probabilities greater than 1/2, the symbol's Huffman codeword is one bit. Encoding outcomes of self-information 0.5 or less with one-bit codewords is very inefficient. Prefix codes are most inefficient for probabilities greater than 0.64; the greater the probability the worse the coding efficiency. The most inefficient symbol is also the one that occurs most frequently.

To reduce the inefficiency in the single-context case, the strategy is to use run-length coding. The Golomb code is the most efficient prefix code for runlength coding, when the probability of continuing the run is a stationary probability greater than about 0.64. If the probability the run contin-

ues is between 0.5 and 0.64 then a "1" for "not run symbol" and "0" for "run symbol" (or vice-versa) is optimal relative to prefix codes.

Until recently, there was no well-known way to handle context-dependent run-length coding while switching contexts in the middle of a run. (It works if the file is read backwards but the technique hadn't been widely publicised).

The adaptive Huffman code, to date, is relatively slow compared to adaptive arithmetic codes. There is some relatively recent work directed toward speeding up adaptive Huffman coding.

Huffman coding is very competitive in static situations where the alphabet is relatively large and/or where the most probable symbol does not fall into an "inefficiency gap" between probability $1/2$ and $1/4$, or $1/4$ and $1/8$.

Adaptive arithmetic codes tend to be slow in software. However, IBM produces an arithmetic encoder-decoder chip called ABIC with the Q-Coder algorithm, an adaptive binary arithmetic coder, that uses as many cycles as the sum of the input bits plus output bits. The cycle time is about 70 or 80 nanoseconds. The chip also has the neighborhood template context model for bilevel image compression.

In general, Huffman codes favor larger alphabets, where no symbol has a probability above, say $1/8$. Arithmetic codes can better approximate the symbols whose probabilities are above $1/8$. An advantage of Huffman codes is that it simplifies the task of decoding; simple look-up tables can be used.

Lempel-Ziv coding (also called Ziv-Lempel coding)

Both arithmetic codes and Huffman codes are capable of doing symbolwise encoding (one symbol at a time) and can handle multiple context situations on a symbolwise basis. In contrast, the Ziv-Lempel class of codes encode several symbols in a single coding operation. In a sense, these codes dynamically build a dictionary. There are two main versions of the algorithm, called LZ77 (or LZ1) and LZ78 (or LZ2). In LZ1, the dictionary is the last K symbols encoded (decoded) where K is usually 1024, 2048, or 4096 bytes. Coding is done by a codeword consisting of a pointer and length into the dictionary. The codeword identifies the longest subsequence of symbols in the input buffer that appears in the dictionary.

In LZ2, the algorithm (in effect) creates a "parse tree" from the symbols already encoded (decoded), adding one phrase to the tree following each encoding (decoding). LZ2 searches the tree for the longest phrase matching the unencoded input string, and encodes the index of that phrase. Both encoder and

decoder use the same method to assign the next available index to the newly created phrase. The new phrase is the just encoded phrase followed by the first symbol of the next phrase.

These LZ algorithms, in a sense, encode using a statistical model equivalent to the following. The first symbol of the encoded subsequence (for LZ1) or phrase (for LZ2 case) is coded under the zero-order (memoryless) context model. The context for the second symbol is the first symbol, the context for the third symbol is the first two symbols, and so on.

The great advantage of the LZ algorithms is that they capture local context while being relatively fast. In each case, the decoder is relatively simple. In a sense, the context model is chosen as the "parsing model", once the decision to use LZ77 or LZ88 based algorithms.

Patent issues

The earliest arithmetic codes have public domain versions, eg the dissertation by Richard Pasco, the paper by Christopher Jones, and the paper by Witten, Neal, and Cleary. Several of the early ideas at IBM were published in the IBM Technical disclosure bulletin. An IBM patent for word-based coding places in the public domain an interesting arithmetic code [goe87], since none of the claims apply to the code used in the preferred embodiment. Some of the IBM patents can be avoided.

One sticky situation is where the patent office granted AT&T a patent on the public domain carry-over control method described in Jones []. The patent office also granted AT&T a patent on a probability estimation method using count ratios described in [lr81].

The LZ88 algorithm seems to be locked up with a Unisys and an IBM patent, and apparently the vendor pays a royalty when Unix compress is incorporated in the Unix operating system. For modem compression using a compliant Vbis international standard, Unisys licenses LZW (LZ78-based) for a very reasonable one-time charge of \$25,000. Using the algorithm outside the compliant standard, however, requires negotiation to avoid paying a percentage on each sale. IBM is also known to offer reasonable licensing terms.

The LZ77 approach has patents that are avoidable, for example, by the public domain algorithm known as gzip. The LZ algorithms are most popular with text, but can also be used on images. Stac Electronics controls the most critical patent; it purchased a patent from Ferranti that seems to cover most hashing approaches. A viable variant of LZ77, reported on by Fiala and Greene, is also patented. IBM recently announced a very fast highly parallel LZ77-based hardware chip that is used in disk file systems. The algorithm is also patented.

Summary of major introductory points

The more effective lossless compression schemes of any kind depends on (1) proper creation of, and (2) estimation of, the probability distributions. The intellectual activity that creates context-dependent distributions is critical. The contribution of the strictly coding component of compression, as defined here (eg Huffman codes, arithmetic codes, and the coding aspect of the LZ codes), is to provide at best, 100% coding efficiency.

A key viewpoint of context modeling is that theoretically it either breaks even or increases the compression, whereas the (pure or statistical) coding part either breaks even (when 100% efficient) or reduces the compression (when less than 100% efficiency). However a 100% efficient coding approach cannot make up for using the wrong probability distribution relative to the data file.

The number of contexts in a compression algorithm influences the total number of coding parameters, the other factor being, as in [tlr85], the number of coding parameters per distribution. In [tlr85], a hybrid context model approach was taken. The estimation or transmission of the probabilities represents an overhead that tends to limit the number of contexts, because of the overhead of estimating probabilities on the fly, or transmitting them to the decoder, or using sub-optimal "average" statistics in the application.

In [tlr85] for example, the idea is to "manage" the number of coding parameters: first encode the number identifying the quantized range, called the bucket number, of a prediction error under the full local context based on surrounding prediction error buckets, and then knowing the bucket number, encode the value within that particular bucket under its own context. The decoder reconstructs the error value by summing the lowest value belonging to range of the decoded bucket number and the decoded relative value within that range.

Three popular methods of coding include arithmetic and Huffman coding were reviewed. Algorithms that compress data where the context, estimation, and coding are embodied in a single algorithm are the LZ77 and LZ78 families.

Types of compression

- * Lossless
- * Lossy by choosing not to code some information

Lossless Image Compression is also called reversible, exact, perfect reconstruction, or noiseless. In a lossy algorithm, the "noise" is introduced into the image by throwing away information that the human vision system (HVS) will not see anyway. Near-lossless algorithms are sometimes called

"perceptually transparent".

The term "noisy" or "noiseless" is confusing because of its ambiguous meaning. Shannon used the term "noise" to mean the alteration of the original data during transmission, with the consequence that the received data was not the data that was sent. The second usage is due to the so-called quantization noise when intentionally throwing away information.

The lossy algorithms typically have three steps; (1) model the data in a way that facilitates discarding the part of the data that will least disturb the human viewer or listener, and (2) losslessly encode what remains of the data. In many cases, a decade or two ago, the remaining data was encoded using a fixed number of bits that did not take advantage of the statistics. In terms of a histogram of frequencies, the earlier techniques assumed every possible outcome was equally likely and assigned (or allocated) the same number of bits to each outcome.

In the case of subband coding, for example, the bit-allocation problem concerns how many bits to allocate to the spectral coefficients of each band. If a band is allocated two bits, for example, then that band has four quantization levels, each identified as 00, 01, 10, or 11.

Application-dependent and resource-dependent lossless compression approaches

One can have lossless coding for various purposes, depending on the resources available and the specific compression application. In some cases, the available resources may dictate the amount of compression achieved. In other cases, the compression application may require the compressed image retain a particular feature or possess a particular property. Some constraints may include the absence of time or equipment to store the image and employ a compute-intensive algorithm, eliminating the possibility of first determining the statistical properties, encoding them in a header, so that the decoder has a code specific to the image's statistics. Another application may require a lossy version first with a provision for supplying the lossless version.

Popular lossy techniques that have lossless versions: transform-based

Transform-based approaches include Discrete Cosine Transform (DCT), Walsh-Hadamard transform, and others, but also include wavelets and subband coding because each of these approaches convert the original pixel intensity values in the space domain to spectral component values in the spatial frequency domain. This conversion can be viewed as linear algebra involving the multiply/add function of the inner product. (Several workstation architectures now do the multiply/add operation in a single clock cycle.) This manipulation has the effect of converting, in the case of the 8x8 block DCT for example, 64 8-bit pixel integer values to 64 11-bit spectral component integer values. The effect is to create more information to be compressed than

there was to begin with. Although they perform well on near-lossy applications, for lossless applications there must also be some other advantage to using a transform.

The approach that includes wavelets and subband coding manipulate the data in a lossless manner called "perfect reconstruction". These approaches employ linear algebra. Subband coding can be set up as linear operations or convolutions (weighted sums of pixel values on the forward side). With proper selection of the forward and inverse operations, subband coding can be lossless. Unlike the DCT, the convolutions employed may have no more than 12 to 16 pixel values affecting each component, and so the increase in information due to increase in range of the component values is not as severe as the 8x8 DCT.

Wavelets and subband coding treat the low frequency (smooth) parts and the high-frequency (edge information) into "bands" by applying to both dimensions the convolutions or transforms that split the pixel values into their frequency components. The "low band" components are actually smoothed pixel values. The number of components is equal to the number of pixel values, so the lowest frequency value for a 4x4 image block should, in the lossless case, be the sum of the 16 pixel values in the block (or at least have the same number of bits).

The wavelet and majority of subband approaches (wavelet being a "subset" of subband coding) use a high-low split in each direction (first the horizontal (X) axis and next the vertical (Y) axis, that converts each 2x2 square of values into four components traditionally denoted LL, LH, HL, and HH. Band LL (Low-Low) is the result applying the smoothing convolution first along the X-axis (horizontally) and then along the Y axis (vertically).

In both the wavelet and subband coding approaches, the LL band components are regrouped as their own block in the upper left corner of the image or sub-image they are in, the LH in the upper right, HL in the lower left, and HH in the lower right. Wavelets and subband use a "pyramid" approach where the LL band is re-split into four components. In the wavelet technique, the low-band is re-split until there's nothing more to split: the re-splitting stops when the upper left hand pixel position of the compressed image contains the component representing the average intensity value for the entire images.

Model-based, feature-based, or Second-generation image coding

M. Kunt and colleagues [3] applied the term "second-generation" to image coding techniques that incorporate some image analysis functions, in a lossy fashion, prior to the lossless coding step. A typical example of a second-generation technique will first segment the image into regions, using any number of segmentation techniques. The regions represent areas of more or less homogeneous range of pixel values, and the region boundaries most often

capture the edge information. The coding is done in a two-step process. First the region boundaries are encoded, and second the interior of the region is encoded. The simplest description of an interior is the mean value.

A more complicated description of a region interior ensues by viewing the pixel intensity value a 3rd dimension or axis z , above the pixel-locations's 2-dimensional plane in the x and y axes. More complicated descriptions include fitting a surface, in the pixel-intensity dimension, "above" the region.

The first-generation techniques, then, are "pixel-based". The pixel-based approaches operate directly on the pixel values, and generally do not attempt to detect some of the higher-level features of the image. In a way, the use of local contexts capture some of the low-level features. However, the segmentation-based approaches cover a much larger scale of activity. Context-based approaches are not on the same scale as the "second generation" approaches. The fact that context model based image compression are first generation is particularly true when only three or four neighboring pixels take place in the context.

Lossless techniques by encoding the residual of a lossy technique

Inherently lossy algorithms, such as Vector Quantization (VQ), can be made lossless by encoding the so-called residual: the difference between the result of the lossy algorithm and the original image. This lossy + residual is being studied by NASA for distributing satellite images of Earth to scientists over internet. The scientists workstation has decompression software. To browse an image database, the scientist receives and decompresses a small (lossy) image before requesting the residual.

Hierarchical approaches

The wavelet and subband approaches use convolution (weighted sum) techniques to separate the smooth from the detail, while generating a pyramid by re-splitting the LL band. The Laplacian pyramid approach is similar in spirit in that the "low band" spectral value covers a larger and larger area. The higher up the pyramid, the fewer the low band values at that resolution, and therefore the resolution of the image is lower, hence each low-band value lies above a larger area of the original (the highest resolution) image.

The general idea can be explained most simply using the quadtree structure. Replace each quad (four-pixel 2x2 block) by the sum of the pixel values (which is 4 times the mean). Alternatively, replace the sum with the actual mean: the sum divided by 4, retaining the fractional part. The 2x2 block can be reconstructed from the mean if we remember the difference value of three pixels from the mean. In a sense, we can use the mean as the "prediction" for each of the four pixel values, and encode the differences. However, if we know the

mean and reconstruct three of the four lower-level pixel values, the fourth pixel location value is known to be the difference between four times the mean value (minuend) and the sum of the three known pixel values (subtrahend). So what appears to be five unknowns (the block mean and the four differences) can be represented by four unknowns (the block mean and three of the differences). Again (as with transform-based approaches) we encounter the need for more than 8 bits per unknown when the original pixel values are 8-bit values between 0 and 255.

The hierarchical approach is popular for "progressive transmission" applications. Progressive transmission is motivated by the notion that it is "bad taste" to keep the viewer waiting too long for the image to be thrown up on the screen. Therefore, if the compression is slow, data retrieval is slow, or the transmission line is slow, give the viewer an early sketch of the image and progressively build the picture as it arrives. In the video conferencing arena, a supplier has a competitive advantage by using progressive build-up of the image.

Hierarchical and transform approaches obligate the compression algorithm to compress more bits than are in the original image

Consider 8-bit pixels. In the above procedure, we have replaced four 8-bit values a mean with a 10-bit value: the integer part of the mean has 8 integer bits, and 2 fractional bits. The difference values carry 2 fractional bits, and we hope most of the higher-order integral bits are zero.

Transforming the original set of 8-bit pixel values and ending up with more bits to encode than we started with is critical in lossless coding. The original idea of transforming the original data is to better quantize out (discard) information the human will not see. The main reason for a transform disappears for lossless coding, since when everything must be preserved there is no longer any reason for transforming the problem to a domain suited for quantizing away what the viewer will not see.

For lossless compression, the only justification for transforming the data is a reduction in the number of bits needed to encode the image. And since the compression depends only on statistical considerations, the best lossless compression comes from the best statistical approach.

Predictive coding offers a favorably re-shaped probability distribution without increasing the number of bits to be compressed.

For pixel-based (first generation) lossless image compression techniques, we favor context-dependent predictive coding as providing the best compression for a given number of coding parameters. This statement becomes more true (intuitively) with increasing coding parameters (until the overhead takes

over). We feel, therefore, that context-dependent predictive coding for images provides the best scenario for making cost-performance tradeoffs given the requirement the compression algorithm be lossy. The predictive coding approach is flexible in that it allows for both one-pass and two-pass algorithms.

For given applications, the compression system designer can craft the contexts, and how to decompose the prediction error in a flexible fashion to meet other applications requirements such as speed, or simplicity of implementation, or other factors that bear on the intended market.

Predictive coding applies the equation $V = P + E$ to reconstruct original value V from prediction P and error E . The error E is calculated as $E = V - P$. Consider again, 8-bit pixel values that range from 0 to 255. For given P (a value known both to the encoder and to the decoder (it is calculated from previously respectively encoded or decoded values), there are only 256 possible error values E (since P is fixed and there are only 256 values that V takes on). Thus, instead of requiring 9 bits to specify a value E between -255 and +255, the predictive coding problem of "E given P" requires only an 8-bit error value E . That value specifies which of the 256 possible error values will reconstruct original value V .

We call this 8-bit number the Bostelmann number after the person who first published the idea. The arithmetic is fairly simple. If V ranges from 0 to $R-1$, then using modulo R addition and subtraction in the "2s complement" number system does the trick. Assume the arithmetic unit employs the 2s complement number system for representing negative numbers (which most if not all current machines tend to use). For 8-bit values V , this simply means you keep the 8 least significant bits of calculation $E = V - P$ at the encoder, and that -following sign extension- perform $V = P + E$ with 2s complement arithmetic at the decoder. The least significant 8 bits of this result represents the original value V between 0 and 255.

Review and comment on lossless image compression art

Bilevel images

Bilevel images are transmitted over facsimile machines. Early fax equipment used analog signals without compression. Current fax machines digitize the scanned document in raster-scan order, convert the signal to binary (0 is white and 1 is black) and employ one of the international standards developed by the CCITT.

A review of early lossless bilevel image compression work appears in [arps]. Since black areas and white areas are found together, many early approaches encoded a white run followed by a black run. The main alternative is called

contour coding, where the contours are chain-coded together.

The algorithms employed in current facsimile systems are the standards called G3 and G4 (Group 3 and Group 4). G3 uses a combination of the earlier one-dimensional run-length codes together with a two-dimensional feature called READ (Relative Element Address Designate) that describes where the current run ends relative to where the corresponding run on the line above ended. The algorithm itself for G3 is called MR (Modified Read), and for G4 is MMR (Modified Modified Read).

The state-of-the art in bilevel image compression is the JBIG international standard. The basic approach of the non-hierarchical version is that of [bw], which is a one-pass algorithm. It's context model is the neighborhood template (pixel values in the causal plane, i.e., already decoded) whose bit values, concatenated together, form an address to the binary statistic for each particular context. Coding is done using adaptive binary arithmetic coding. In [lr81], the Laplacian estimator (applicable to stationary statistics) was compared with two algorithms (SS-adap and SSS-adap) applicable to non-stationary statistics.

In JBIG, the coder is a 10-pixel neighborhood, and the adaptive binary arithmetic code is called the QM-coder. The QM-coder uses the adapter-coder technique (patented by IBM) of the Q-coder, see eg [pm93]. This technique uses the renormalization inherent in the arithmetic coding process to drive the adaptation algorithm. The primary beneficiary of this technique is the improved speed of the software implementations: one need not involve the probability estimator until one or more bits of the code string need to be shifted out of the coding registers.

Gray-scale images

Most lossless compression algorithms use predictive coding, where the predictor function uses the values of nearby pixels already encoded (decoded) to produce a prediction for the next value to be encoded or decoded. With respect to the location of the next pixel, popular locations used for the predictor function include the W, NE, N, and NW (west, northeast, north, and northwest) locations.

Rice's code for lossless image compression

A very early lossless image compression algorithm, called the Rice Machine [rp71], employs DPCM (where the prediction is the value of the W pixel location). The prediction errors were encoded in blocks of, say, 16 pixels. There were several ways of coding the prediction errors in the block, depending on the block statistics.

The Rice code has evolved over the years. A basic idea is to convert

the positive and negative error values into a single sequence, called the Fundamental Sequence (FS), by doubling the absolute value of the error, and subtracting 1 if the error is negative. The sequence of values FS goes 0, -1, 1, -2, etc.

In the most recent version, the error value is converted to a "run-length" by identifying the position of the error in the FS. A binary string is formed of the run-length type, i.e., a string of the form 00..01. All error value positions not equal to the current error value are zero, and the error value position is 1. For example, error 0 converts to 1, corresponding to a runlength of 0. Error-1 converts to a runlength of 1, which is 01. Error +2 converts to 00001. Note that +2 is the fifth position in the FS, and 00001 represents a run-length of 4.

The runlength code can be statistically modeled by the binary probability $p(0)$, the probability the run continues to the next higher value. Golomb has devised an optimal code for encoding runlengths, which is called the Golomb code. Golomb's parameter "m" is determined as the integral power of $p(0)$ such that $p(0)^m$ is close to $1/2$.

The current version of Rice's code uses values of m that are powers of 2 (1, 2, 4, 8, etc), and this value is determined for each block of 16 prediction errors. The value m is identified in the block header, which is followed by 16 prefix codewords, one for each of the 16 errors in the block.

This incorporates the local context by blocking 16 prediction errors into the most favorable statistic. The encoder "looks ahead" at the next 16 values, calculates the best Golomb number m for the code, and passes the value of m to the decoder as so-called side information.

General linear prediction (more complex than DPCM)

A linear predictor is a weighted sum of local pixel values, where the weights sum to unity. In general, if the weights are positive and sum to unity there is no "increase" in the noise (hence accuracy) of the predictor. Of the predictors tested, $0.5W + .25N + .25NE$, or $0.25W + .5N + .25NW$, seem to perform the best.

Nonlinear predictors have also been successfully used. The nonlinear predictors that are most popular use a median value.

Many studies continue to address the problem of the best predictor weights, although recent work has demonstrated that use of contexts improves the compression much more than designing the predictor function. Most of these

mis-addressed studies assume that the prediction error statistics represent a memoryless source, rather than being influenced by local statistics as in the earlier successful Rice approach. Even the quality metric for the optimum predictor is wrong: rather than use minimum codelength as the optimization criteria, these investigations minimize the variance of the error.

Context-dependent lossless image compression

A 1980 IBM internal conference paper [lrt80] shows that with context-dependent treatment of the prediction error distributions, i.e., splitting the errors into independent distributions, offers greater compression than fiddling with the predictor weights. The approach uses quantized prediction errors from local pixel location already encoded as the context. Quantizing the prediction error at each location serves as a context component. Each quantization range is called a "bucket". Using the ordered bucket values from a set of neighbors reduces the number of contexts over using the pixel values themselves or error values themselves as the context. The published result appears as [tlr85].

In [lm81], Lehmann and Macovski designed four Huffman codes for the prediction error, using as a local context the error variance of neighboring pixel values. In [ghar84], Gharavi encodes context-dependent prediction errors with a set of Huffman codes.

In 1984, Langdon [lan88] implemented the idea of [tlr85] in an algorithm called Sunset. The quantization of prediction errors used a power-of-two scale so the encoder and decoder just needed to identify the most significant bit of the error to determine the context. The Sunset approach uses an adaptive binary arithmetic coder. The binary outcome of zero/non-zero is encoded under full context, and (if nonzero) so is the sign. The coder also encodes the context as a separate statistical event, and decomposes the outcome via a "logarithmic search" binary decision tree. The extrabits are individually coded in their own context, as described in [tlr85].

The lossless JPEG compression algorithm is an independent algorithm from the lossy JPEG algorithm, in that lossless JPEG does not use the DCT transform. The independent JPEG lossless algorithm that uses the binary arithmetic coder known as the QM-coder, although not designed by Langdon, is easily described in terms of the differences with Sunset. The adaptive binary coder of Sunset is replaced by the QM-coder. Value 1 is subtracted from the prediction error if it is not zero, and the sign is also encoded separately. The binary tree used by JPEG lossless is instead a "sequential search" decision tree. The contexts are slightly different, and fewer in number than Sunset. The extrabits use their own context plus the context of whether the pixel above had a large error or not. See [pm93] sec 10.3, for a description of the lossless JPEG algorithm.

In [lgs92], Langdon and colleagues describe an algorithm called SunJPEG that combines into Sunset some of the features of lossless JPEG. This study has continued, the latest version appearing in [lh94]. The program called cb6.c by Langdon and Haidinyak was used in a comparison request by Roz Picard of MIT. The interaction of these algorithms is attached, with results from the two-pass program for "exaggerated consensus" (EC-2P) [pp94], and the SunJPEG based cb6, and Weinberger's program DCXT-BT/CARP [wra94]. The comparison of these algorithms is done on the same set of test files from MIT, provided by Roz Pickard.

Lossless image compression can also be performed by the JBIG standard. A comparison is made in [at94] between lossless gray-scale compression with JPEG, and bit-plane encoded JBIG. For a relatively small number of bits per pixel gray scale, the JBIG approach wins. For six or more bits per pixel, the arithmetically encoded JPEG lossless wins.

Medical images - must be reversible

For legal reasons, medical images used in diagnosis tend to require lossless compression. A widely referenced work [rvvp88] compared their method called Hierarchical Interpolation, or HINT, against some other methods, for lossless compression. In HINT, the image is first subsampled as the corners of a 4x4 block, to form the lowest resolution image. Sampling is from upper left to lower right, so the "next pixel location" in the image uses the previously sampled pixel values in the N, NW, and W positions as a prediction. The method is lossless so the error is encoded. At the lowest resolution, these pixels are 4 locations away from the next pixel. The next step fills in the center pixel; i.e., the pixel location lying 3 pixels away along the NE to SW and the NW to SE diagonals. This is done by interpolation, and encoding the error. The next interpolation uses these new points plus the old to fill in pixels in the north-south and east-west directions that are 2 pixel locations away. The N-S and E-W interpolations are followed by the diagonal interpolations and so on. These distinct interpolations can be encoded under a distribution for each level. As the resolution improves, the error distribution becomes more peaked about the zero error value.

In the competitive procedures, the paper did not have as much statistical context brought to bear, and [rvvp88] was declared the winner. Since then, others have compared their own ideas against HINT, see [db93] [hv91] [kr92] [rc92].

Double-adaptation techniques (one pass)

In file compression, a technique called double-adaptation is described in [lr83]. In this technique, the compression takes place in one pass. In the

single pass, the algorithm begins compression in the zero-order model state. However, it adaptively adds higher-order contexts (first order and a special second-order 'runlength' context) as their popularity becomes known. Within each context, probability estimation techniques are employed to determine the coding probabilities. Since 1983, others have taken the idea in various directions. A treatment of the subject appears in book form [wil91].

Adaptive contexts for images first appears in [rc92], where the method of double-adaptation is extended to images, and by generating contexts on a binary basis.

Weinberger, Rissanen, and Arps have applied double adaptation, using a context growing and using algorithm simply called "context" by Rissanen [ris83]. The idea has been furthered in [wra94].

Two-pass technique of "clustering"

Clustering is first described in [mr74], following the filing of several patents on Huffman coding and compression. The idea is to collect count frequencies for a higher-order context model, and then merge context states that have similar statistics. This is one way to reduce the number of coding parameters. In [lst92], the idea is applied to images. A recent use of clustering, called exaggerated consensus, is by Kris Popat and Roz Pickard [pp94].

FELICS

A simple-to-implement lossless image compression, which give surprisingly good compression given its simplicity, is reported in [hv93]. The method uses prefix coding, and codes pixel values relative to the range described by the value and location W and value in location N. A 1-bit code describes if the pixel is in this center range, if not a 1-bit code describes which side (above or below) the value lies. The code for the center range is a function of the number of values lying in the range, and the code for the above and below ranges is symmetrical with respect from the distance to the center range.

Use of LZW in lossless image compression

A paper by Lewis and Forsyth [lf90] uses LZW to compress the prediction errors for a lossy DPCM algorithm, as well as lossless. The algorithm could similarly be used in the context-dependent lossless case.

Summary

Following an overview of compression in general, including a description of the relative importance of context modeling, statistics determination (estimation)

and "pure" coding (distinguished from model-code intertwined), we provide a separate conclusions section. Then we overview the major approaches to lossless image compression in the bilevel and gray-level domains.

For a relatively small number of bits per pixel, the gray-level image compresses better under a bilevel image compression algorithm applied to the bit-planes. Bilevel algorithms use adaptive binary arithmetic coding and the neighborhood template in the JBIG standard.

For gray-scale images, the strength in the pixel-based compression scheme seems to lie in surrounding the next pixel to be compressed with a useful context. Predictive coding seems to be the winner here. Second-generation techniques offer the promise of increased compression with greater complexity but this approach has not been as thoroughly explored as the context-dependent pixel-oriented approaches.

The second-generation techniques, to be applicable to lossless image compression, would follow the lead of the "encode the residual" approaches. The idea would be to keep the residual as small as possible, and then try the context-dependent pixel-based approach on those differences.

The pixel-based approaches that do not bring the closer or more local pixel values to bear on the problem, such as HINT, empirically seem inferior.

REFERENCES

- [arps79] R. B. Arps, "Binary image compression", (book chapter) in Image Transmission Techniques, Advances in Electronics and Electron Physics, vol 12, W. K. Pratt, Ed., Academic Press, New York, 1979.
- [at94] R. Arps and T. Truong, "Comparison of International Standards for Lossless Still Image Compression", Proc IEEE, v 82, no 6, June 1994, 889-899.
- [ba83] P. Burt and E. Adelson, "The Laplacian pyramid as a compact image code", IEEE Trans Commun, vol COM-31, no 4, pp 532-540, Apr 1983.
- [db93] M. Das and S. Burgett, "Lossless Compression of Medical Images Using Two-Dimensional Multiplicative Autoregression Models", IEEE Transactions on Medical Imaging, vol 12, no 4, December 1993, 721-726.
- [fal73] Newton Faller, "An Adaptive System for Data Compression", Conference Record, Seventh Asilomar Conference on Circuits, Systems, and Computers, IEEE, Nov 1973, 593-597.

- [gal78] R. Gallager, "Variations on a theme by Huffman", IEEE Trans Inf. Theory, vol IT-24, no 6, November 1978, 668-674.
- [ghar84] H. Gharavi, "Conditional Variable-Length Coding for Grey Level Pictures", AT&T Bell Systems Technical Journal, v 63, n 2, Feb 1984, 249-259.
- [goe]87] G. Goertzel, "File Compressor", US Patent 4,672,539, June 9, 1987.
- [hv91] P. Howard and J. Vitter, "New Methods for Lossless Image Compression using arithmetic coding", Proc Data Compression Conference, Snowbird UT, Apr 8-11, 1991, 257-266.
- [hv93] P. Howard and J. Vitter, "Fast and Efficient Lossless Image Compression", Proc Data Compression Conference, IEEE Computer Society, Snowbird UT, Mar 30 - Apr 2, 1993, 351-360.
- [kr92] G. Kuduvali and R. Rangayyan, "Performance Analysis of Reversible Image Compression Techiques for High-Resolution Digital Teleradiology", IEEE Trans Medical Imaging, vol 11, no 3, September 1992, 430-445.
- [kun95] M. Kunt, "Second-generation image-coding techniques", Proc IEEE, vol 73, no 4, pp 549-574, Apr 1985.
- [lan88] G. G. Langdon, Jr., "Further Developments in Lossless Gray-scale Image Compression", IBM Research Report RJ-6426, September 8, 1988. Text from paper and talk at IBM ITL on Pattern Recognition and Image Processing, Yorktown Heights NY, November 1984.
- [lan91] G. Langdon, "Sunset: a hardware-oriented algorithm for lossless compression of gray scale images", Proc SPIE, (Image Capture, Formatting, and Display) vol 1444, February 1991, 272-282.
- [lgs92] G. Langdon, A. Gulati, and E. Seiler, "On the JPEG Model for Lossless Image Compression", Proc Data Compression Conference, IEEE Computer Society, Snowbird UT, Mar 24-27, 1992, 172-180.
- [lf90] H. Lewis and W. Forsyth, "Hybrid LZW Compression", Proc SPIE, vol 1244 (Image Processing Algorithms and Techniques), 182-189, 1990.
- [lh94] G. Langdon and C. Haidinyak, "Context-dependent distribution shaping and parameterization for lossless image compression", Proc SPIE vol 2298 (Applications of Digital Image Processing XVII), July 1994.
- [lko94] Heesub Lee, Yongmin Kim, and Seho Oh, "Lossless compression of medical images by prediction and classification", Optical Engineering, vol 31, no 1,

January 1994, 160-166.

[lm80] L. Lehmann and A. Macovski, "Data compression of x-ray images by adaptive DPCM coding", Proc SPIE (Conference on Digital Radiology), vol 314, Stanford University, September 1981, 396-404.

[lr81] G. Langdon and J. Rissanen, "Compression of Black-white images with arithmetic coding", IEEE Trans commm., vol COM-29, no 6, 858-867, June 1981.

[lrt80] G. G. Langdon, Jr, J. Rissanen, and S. Todd, "On Lossless Compression of Gray-scale Images", IBM Research Report RJ 8028, March 18, 1991. Declassified proceedings paper of IBM Internal Conference on Data Compression of March 1980.

[lst92] S-M Lei, M-T Sun, and K-H Tzou, "Design and Hardware Architecture of High-Order conditional entropy codeing for images", IEEE Trans. Video Technology, vol 2, no 2, pp 176-186, June 1992.

[lr83] G. Langdon and J. Rissanen, ``A Double-adaptive File Compression Algorithm'', IEEE Trans. on Communications, vol COMM-31, No. 11, November 1983, 1253-1255.

[lst92] S-H Lei, M-T Sun, and K-H Tzou, "Design and Hardware Architecture of High-Order Conditional Entropy for Images", IEEE Trans on C&S for Video Technology, vol 2, no 2, June 1992, 176-186.

[mr74] J. Mommens and J. Raviv, "Coding for Data Compaction", IBM Research Division, Report RC5150, Yorktown Heights NY, November 1974.

[mr89] P. Melnychuck and M. Rabbani, "Survey of lossless image coding techniques", Proc SPIE, vol 1075, (Digital Image Processing Applications), pp 92 - 100, 1989.

[pm93] W. Pennebaker and J. Mitchell, JPEG: still image data compression standard, Van Nostrand Reinhold, New York, 1993. Section 10.3, pp 162-185.

[pp95] K. Popat and Roz Picard, "Exaggerated Consensus in lossless image compression", Proc IEEE First International Conference on Image Processing, Austin TX, November 1994.

[rc92] T. V. Ramabadran, and Keshi Chen, "The Use of Contextual Information in the Reversible Compression of Medical Images", IEEE Trans Medical Imaging, v 11, no 2, June 1992.

[ris83] J. Rissanen, "A Universal Data Compression System", IEEE Trans Inf

Theory, vol IT-29, no 5, 656-664, Sep 1983.

[rl81] J. Rissanen and G. Langdon, "Universal Modeling and Coding", IEEE Trns Inf Theory, vol IT-27, 12-21, Jan 1981.

[rj91] M. Rabbani and P. Jones, "Digital Image Compression Techniques", SPIE Technical Publication (Tutorial) book, vol TT07, 1991.

[rp71] Robert F. Rice and James R. Plaunt, "Adaptive Variable-length Coding for Efficient Compression of Spacecraft Television Data", IEEE Trans Commun., vol COM-19, no 6, December 1971, 889-897.

[rvvp88] P. Roos, M. Viergever, M. Van Duke, and J. Peters, "Reversible Interframe compression of medical images", IEEE Trans Medical Imaging, vol 7, 328-336, Dec 1988.

[shah91] I. A. Shah, "A Chip Set for Lossless Image Compression", IEEE JI of Solid-State Circuits, vol 26, no 3, March 1991, 237-244.

[tlr85] Stephen Todd, Glen G. Langdon, Jr., and Jorma Rissanen, "Parameter Reduction and Context Selection for Compression of Gray-scale Images", IBM JI Research and Development, vol 29, no 2, March 1985, 188-193.

[wil91] Ross Williams, "Adaptive Data Compression", Kluwer Academic Pub, New York, 1991.

[wra94] Marcelo Weinberger, Jorma Rissanen and Ronald Arps, "On Universal Context Modeling for Lossless Compression of Gray Scale Images", submitted to IEEE Transactions on Image Processing.

Attachment - recent activity in lossless image compression

Based on request for comparisons by Roz Picard of MIT

>From Marcelo Weinberger: ([wra94])

Glen,

Roz Picard asked me to run the version of Context Algorithm I developed with Jorma and Ron on a set of images that she used for her 'Exaggerated Consensus' algorithm (EC-2P). She told me that you did the same with CB6, and she suggested that I share my results with you,

as she was quite impressed. In fact, in the same set of experiments, besides Context (or, rather, a version thereof that I rewrote here, similar to the one with Jorma and Ron), I also tried the version of Sunset that you gave me (is it the same as CB6?). Taking our version of Context as a benchmark, Sunset (CB6?) confirmed its very good performance and seems preferable to EC-2P: the average result I got for Sunset was somewhere between Context and EC-2P, and confirmed the relative performance reported in my paper with Jorma and Ron, so there is probably not much new for you here. Roz suggested that EC-2P performed similarly to CB6, but at least in my experiments Sunset was superior to EC-2P in most images (again, it's probably not the same program). I'm appending the results I sent to her.

TABLE 1. Comparison - 'bits/pixel' on 8-bit pixel value

DCXT/BT-CARP (Difference Context Algorithm with Binary Tree and Conditional AR Prediction).

Image	DCXT-BT/CARP	EC-2P	CB6
-----	-----	-----	---
Al	3.77	4.41	
aero2	4.00	4.73	
b2	4.05	4.23	
baboon (*)	4.97	5.91	6.06
bank.512	4.08	4.32	4.39
cman	4.68	4.73	4.85
couple	2.32	3.54	2.56
crowd	5.94	5.97	6.10
einsteinB	3.90	4.24	
face	4.32	4.59	
fruit (*)	3.80	4.66	
girl.512	3.74	3.92	3.79
girl2k	4.71	4.83	
hat	4.06	4.35	
jet	3.91	4.07	4.00
kids	4.44	4.50	
lenna	3.99	4.20	4.04
loco.512	4.41	4.43	4.62
london	3.24	4.03	3.48
mill	5.08	5.37	5.29
oleh	3.84	4.12	3.87
pyramid	3.00	3.93	3.30
reagan	3.94	4.12	
tek-boat	5.25	5.52	5.47
tek-cute	3.91	4.12	3.97
tek-rose	5.81	5.82	5.98

vegas	3.80	4.23
wed	4.30	4.56

(*) These two are 7 bits/pixel images to which a dummy bit has been appended (just look at the histograms). Hence, when compressed as they are, DCXT-BT/CARP is a bad idea since it assumes a Laplacian model and, consequently, it assigns non-zero probability to events that will never occur. The compression ratios for the original images are 5.98 and 4.81 bits/pixel respectively (notice that there is exactly a whole bit difference: the parametric model couldn't compress the dummy bit!). Something similar is valid for jet, but the eighth bit was assigned in a way that I could not determine, so I compressed the original.