SHORTEN:
Simple lossless and near-lossless waveform compression

Tony Robinson

Technical report CUED/F-INFENG/TR.156

Cambridge University Engineering Department,
Trumpington Street, Cambridge, CB2 1PZ, UK

December 1994

Abstract

This report describes a program that performs compression of waveform files such as audio data. A simple predictive model of the waveform is used followed by Huffman coding of the prediction residuals. This is both fast and near optimal for many commonly occuring waveform signals. This framework is then extended to lossy coding under the conditions of maximising the segmental signal to noise ratio on a per frame basis and coding to a fixed acceptable signal to noise ratio.

1 Introduction

It is common to store digitised waveforms on computers and the resulting files can often consume significant amounts of storage space. General compression algorithms do not perform very well on these files as they fail to take into account the structure of the data and the nature of the signal contained therein. Typically a waveform file will consist of signed 16 bit numbers and there will be significant sample to sample correlation. A compression utility for these file must be reasonably fast, portable, accept data in a most popular formats and give significant compression. This report describes "shorten", a program for the UNIX and DOS environments which aims to meet these requirements.

A significant application of this program is to the problem of compression of speech files for distribution on CDROM. This report starts with a description of this domain, then discusses the two main problems associated with general waveform compression, namely predictive modelling and residual coding. This framework is then extended to lossy coding. Finally, the shorten implementation is described and an appendix details the command line options.

2 Compression for speech corpora

One important use for lossless waveform compression is to compress speech corpora for distribution on CDROM. State of the art speech recognition systems require gigabytes of acoustic data for model estimation which takes many CDROMs to store. Use of compression software both reduces the distribution cost and the number of CDROM changes required to read the complete data set.

The key factors in the design of compression software for speech corpora are that there must be no perceptual degradation in the speech signal and that the decompression routine must be fast and portable.

There has been much research into efficient speech coding techniques and many standards have been established. However, most of this work has been for telephony applications where dedicated hardware can used to perform the coding and where it is important that the resulting system operates at a well defined bit rate. In such applications lossy coding is acceptable and indeed necessary order to guarantee that the system operates at the fixed bit rate.

Similarly there has been much work in design of general purpose lossless compressors for workstation use. Such systems do not guarantee any compression for an arbitrary file, but in general achieve worthwhile compression in reasonable time on general purpose computers.

Speech corpora compression needs some features of both systems. Lossless compression is an advantage as it guarantees there is no perceptual degradation in the speech signal. However, the established compression utilities do not exploit the known structure of the speech signal. Hence `shorten` was written to fill this gap and is now in use in the distribution of CDROMs containing speech databases [1].

The recordings used as examples in section 3 and section 5 are from the TIMIT corpus which is distributed as 16 bit, 16kHz linear PCM samples. This format is in common used for continuous speech recognition research corpora. The recordings were collected using a Sennheiser HMD 414 noise-cancelling head-mounted microphone in low noise conditions. All ten utterances from speaker `fcjf0` are used which amount to a total of 24 seconds or about 384,000 samples.

3 Waveform Modeling

Compression is achieved by building a predictive model of the waveform (a good introduction for speech is Jayant and Noll [2]). An established model for a wide variety of waveforms is that of an autoregressive model, also known as linear predictive coding (LPC). Here the predicted waveform is a linear combination of past samples:

$$\hat{s}(t) = \sum_{i=1}^{p} a_i s(t-i) \tag{1}$$

The coded signal, $e(t)$, is the difference between the estimate of the linear predictor, $\hat{s}(t)$ and the speech signal, $s(t)$.

$$e(t) = s(t) - \hat{s}(t) \quad (2)$$

However, many waveforms of interest are not stationary, that is the best values for the coefficients of the predictor, a_i, vary from one section of the waveform to another. It is often reasonable to assume that the signal is pseudo-stationary, i.e. there exists a time-span over which reasonable values for the linear predictor can be found. Thus the three main stages in the coding process are blocking, predictive modelling, and residual coding.

3.1 Blocking

The time frame over which samples are blocked depends to some extent on the nature of the signal. It is inefficient to block on too short a time scale as this incurs an overhead in the computation and transmission of the prediction parameters. It is also inefficient to use a time scale over which the signal characteristics change appreciably as this will result in a poorer model of the signal. However, in the implementation described below the linear predictor parameters typically take much less information to transmit than the residual signal so the choice of window length is not critical. The default value in the shorten implementation is 256 which results in 16ms frames for a signal sampled at 16 kHz.

Sample interleaved signals are handelled by treating each data stream as independent. Even in cases where there is a known correlation between the streams, such as in stereo audio, the within-channel correlations are often significantly greater than the cross-channel correlations so for lossless or near-lossless coding the exploitation of this additional correlation only results in small additional gains.

A rectangular window is used in preference to any tapering window as the aim is to model just those samples within the block, not the spectral characteristics of the segment surrounding the block. The window length is longer than the block size by the prediction order, which is typically three samples.

3.2 Linear Prediction

Shorten supports two forms of linear prediction: the standard pth order LPC analysis of equation 1; and a restricted form whereby the coefficients are selected from one of four fixed polynomial predictors.

In the case of the general LPC algorithm, the prediction coefficients, a_i, are quantised in accordance with the same Laplacian distribution used for the residual signal and described in section 3.3. The expected number of bits per coefficient is 7 as this was found to be a good tradeoff between modelling accuracy and model storage. The standard Durbin's algorithm for computing the LPC coefficients from the autocorrelation coefficients is used in a incremental way. On each iteration the mean squared value of the prediction residual is calculated and this is used to compute the expected number of bits needed to code the residual signal. This is added to the number of bits needed to code the prediction coefficients and the LPC order is selected to minimise the total. As the computation of the autocorrelation coefficients is the most expensive step in this process, the search for the optimal model order is terminated when the last two models have resulted in a higher bit rate. Whilst it is possible to construct signals that defeat this search procedure, in practice for speech signals it has been found that the occasional use of a lower prediction order results in an insignificant increase in the bit rate and has the additional side effect of requiring less compute to decode.

A restrictive form of the linear predictor has been found to be useful. In this case the prediction coefficients are those specified by fitting a p order polynomial to the last p data points, e.g. a line to the last two points:

$$\begin{aligned} \hat{s}_0(t) &= 0 && (3)\\ \hat{s}_1(t) &= s(t-1) && (4)\\ \hat{s}_2(t) &= 2s(t-1) - s(t-2) && (5)\\ \hat{s}_3(t) &= 3s(t-1) - 3s(t-2) + s(t-3) && (6) \end{aligned}$$

Writing $e_i(t)$ as the error signal from the ith polynomial predictor:

$$\begin{aligned} e_0(t) &= s(t) && (7)\\ e_1(t) &= e_0(t) - e_0(t-1) && (8)\\ e_2(t) &= e_1(t) - e_1(t-1) && (9)\\ e_3(t) &= e_2(t) - e_2(t-1) && (10) \end{aligned}$$

As can be seen from equations 7-10 there is an efficient recursive algorithm for computing the set of polynomial prediction residuals. Each residual term is formed from the difference of the previous order predictors. As each term involves only a few integer additions/subtractions, it is possible to compute all predictors and select the best. Moreover, as the sum of absolute values is linearly related to the variance, this may be used as the basis of predictor selection and so the whole process is cheap to compute as it involves no multiplications.

Figure 1 shows both forms of prediction for a range of maximum predictor orders. The figure shows that first and second order prediction provides a substantial increase in compression and that higher order predictors provide relatively little improvement. The figure also shows that for this example most of the total compression can be obtained using no prediction, that is a zeroth order coder achieved about 48% compression and the best predictor 58%. Hence, for lossless compression it is important not to waste too much compute on the predictor and to to perform the residual coding efficiently.

3.3 Residual Coding

The samples in the prediction residual are now assumed to be uncorrelated and therefore may be coded independently. The problem of residual coding is therefore to find an appropriate form for the probability density function (p.d.f.) of the distribution of residual values

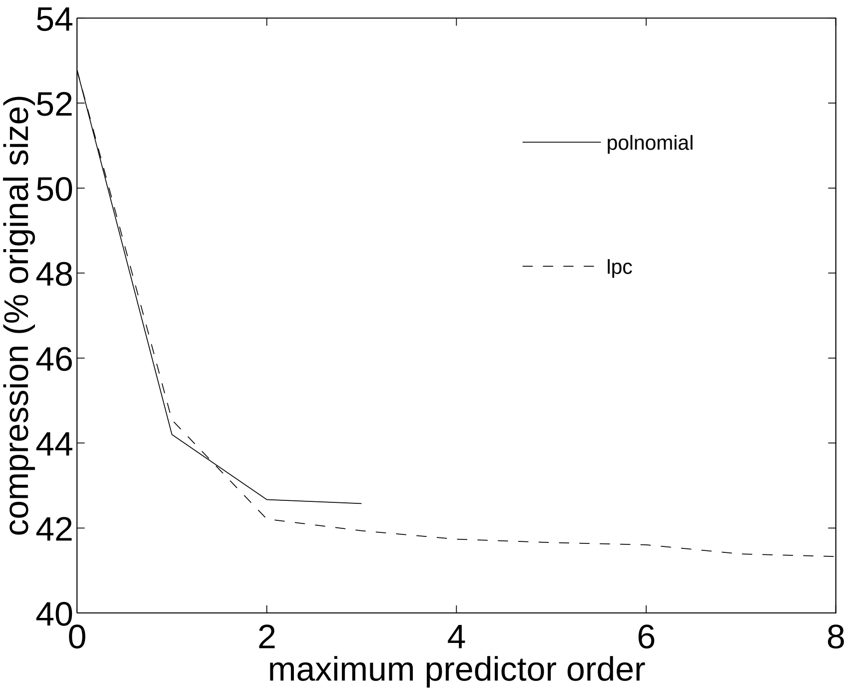

Figure 1: compression against maximum prediction order

so that they can be efficiently modelled. Figures 2 and 3 show the p.d.f. for the segmentally normalized residual of the polynomial predictor (the full linear predictor shows a similar p.d.f). The observed values are shown as open circles, the Gaussian p.d.f. is shown as dot-dash line and the Laplacian, or double sided exponential distribution is shown as a dashed line. These figures demonstrate that the Laplacian p.d.f. fits the observed distribution very well. This is convenient as there is a simple Huffman code for this distribution [3, 4, 5]. To form this code, a number is divided into a sign bit, the nth low order bits and the the remaining high order bits. The high order bits are treated as an integer and this number of 0's are transmitted followed by a terminating 1. The n low order bits then follow, as in the example in table 1.

Number	sign bit	lower bits	number of '0's	full code
0	0	00	1	0001
13	0	01	3	0010001
-7	1	11	2	111001

Table 1: Examples of Huffman codes for $n = 2$

As with all Huffman codes, a whole number of bits are used per sample, resulting in

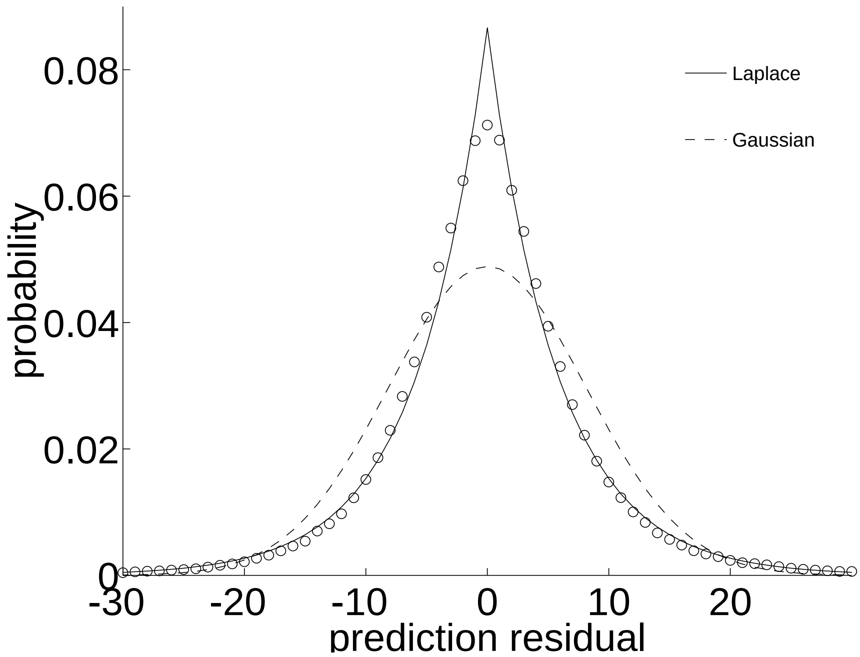

Figure 2: Observed, Gaussian and quantized Laplacian p.d.f.

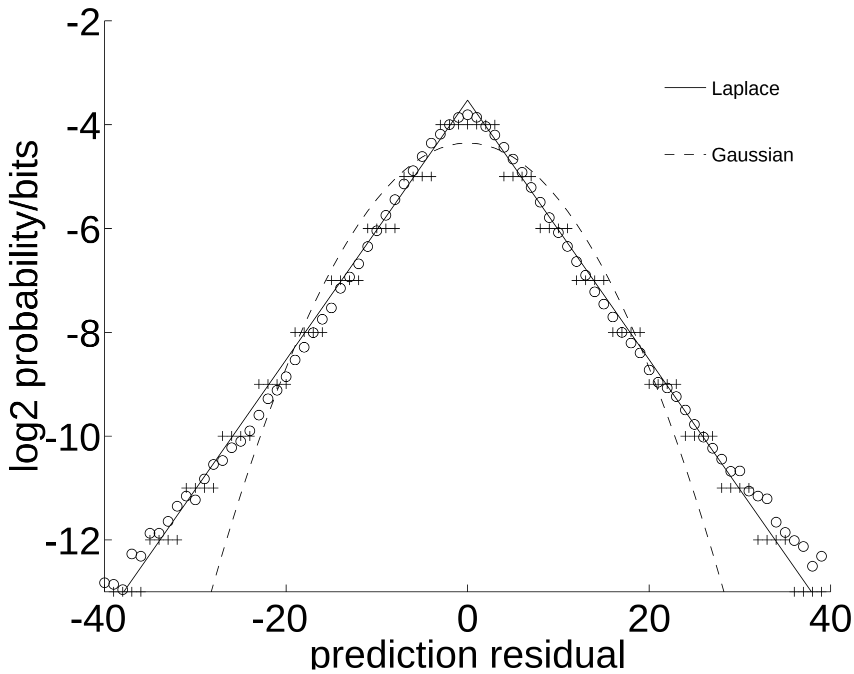

Figure 3: Observed, Gaussian, Laplacian and quantized Laplacian p.d.f and $\log_2$ p.d.f.

instantaneous decoding at the expense of introducing quantization error in the p.d.f. This is illustrated with the points marked '+' in figure 3. In the example, $n = 2$ giving a minimum code length of 4. The error introduced by coding according to the Laplacian p.d.f. instead of the true p.d.f. is only 0.004 bits per sample, and the error introduced by using Huffman codes is only 0.12 bits per sample. These are small compared to a typical code length of 7 for 16 kHz speech corpora.

This Huffman code is also simple in that it may be encoded and decoded with a few logical operations. Thus the implementation need not employ a tree search for decoding, so reducing the computational and storage overheads associated with transmitting a more general p.d.f.

The optimal number of low order bits to be transmitted directly is linearly related to the variance of the signal. The Laplacian is defined as:

$$p(x) = \frac{1}{\sqrt{2}\sigma} e^{\frac{-\sqrt{2}}{\sigma}|x|} \tag{11}$$

where $|x|$ is the absolute value of x and σ^2 is the variance of the distribution. Taking the expectation of $|x|$ gives:

$$\begin{aligned} E(|x|) &= \int_{-\infty}^{\infty} |x| p(x) dx && (12)\\ &= \int_0^{\infty} x \frac{\sqrt{2}}{\sigma} e^{\frac{-\sqrt{2}}{\sigma}x} dx && (13)\\ &= \int_0^{\infty} e^{\frac{-\sqrt{2}}{\sigma}x} dx - \left[x e^{\frac{-\sqrt{2}}{\sigma}x} \right]_0^{\infty} && (14)\\ &= \frac{\sigma}{\sqrt{2}} && (15) \end{aligned}$$

For optimal Huffman coding we need to find the number of low order bits, n, such that such that half the samples lie in the range $\pm 2^n$. This ensures that the Huffman code is $n+1$ bits long with probability 0.5 and $n+k+1$ long with probability $2^{-(n+k)}$, which is optimal.

$$\begin{aligned} 1/2 &= \int_{-2^n}^{2^n} p(x) dx && (16)\\ &= \int_{-2^n}^{2^n} \frac{1}{\sqrt{2}\sigma} e^{\frac{-\sqrt{2}}{\sigma}|x|} dx && (17)\\ &= -e^{\frac{-\sqrt{2}}{\sigma}2^n} + 1 && (18)\\ & && (19) \end{aligned}$$

Solving for n gives:

$$\begin{aligned} n &= \log_2 \left(\log(2) \frac{\sigma}{\sqrt{2}} \right) && (20)\\ &= \log_2 \left(\log(2) E(|x|) \right) && (21) \end{aligned}$$

When polynomial filters are used n is obtained from $E(|x|)$ using equation 21. In the LPC implementation n is derived from σ which is obtained directly from the calculations for predictor coefficients the using the autocorrelation method.

4 Lossy coding

The previous sections have outlined the complete waveform compression algorithm for lossless coding. There are a wide range of applications whereby some loss in waveform accuracy is an acceptable tradeoff in return for better compression. A reasonably clean way to implement this is to dynamically change the quantisation level on a segment-by-segment basis. Not only does this preserve the waveform shape, but the resulting distortion can be easily understood. Assuming that the samples are uniformally distributed within the new quantisation interval of n, then the probability of any one value in this range is $1/n$ and the noise power introduced is i^2 for the lower values that are rounded down and $(n-i)^2$ for those values that are rounded up. Hence the total noise power introduced by the increased quantisation is:

$$\frac{1}{n}\left(\sum_{i=0}^{n/2-1} i^2 + \sum_{i=n/2}^{n-1}(n-i)^2\right) = \frac{1}{12}(n^2+2) \tag{22}$$

It may also be assumed that the signal was uniformally distributed in the original quantisation interval before digitisation, i.e. a quantisation error of $\int_{-1/2}^{1/2} x^2 \mathrm{d}x = 1/12$.

Shorten supports two main types of lossy coding: the case where every segment is coded at the same rate; and the case where the bit rate is dynamically adapted to maintain a specified segmental signal to noise ratio. In the first mode, the variance of the prediction residual of the original waveform is estimated and then the appropriate quantisation performed to limit the bit rate. In areas of the waveform where there are strong sample to sample correlations this results in a relatively high signal to noise ratio, and in areas with little correlation the signal to noise ratio approaches that of the signal power divided by the quantisation noise of equation 22. In the second mode, this equation is used to estimate the greatest additional quantisation that can be performed whilst maintaining a specified segmental signal to noise ratio. In both cases the new quantisation interval, n, is restricted to be a power of two for computational efficiency.

5 Compression Performance

The previous sections have demonstrated that low order linear prediction followed by Huffman coding to the Laplace distribution results in an efficient lossless waveform coder. Table 2 compares this technique to the popular general purpose compression utilities that are available. The table shows that the speech specific compression utility can achieve considerably better compression than more general tools. The compression and decompression speeds are the factors faster than real time when executed on a standard SparcStation I, except the result for the g722 ADPCM compression which was implemented on a SGI Indigo R4400 workstation using the supplied aifccompress/aifcdecompress utilities. The SGI timings were scaled by a factor of 3.9 which was determined by the relative execution times of shorten decompression on the two platforms.

program	% size	compress speed	decompress speed
UNIX compress	74.0	5.1	15.0
UNIX pack	69.8	16.1	8.0
GNU gzip	66.0	2.2	17.2
shorten default (fast)	42.6	13.4	16.1
shorten LPC (slow)	41.7	5.6	8.0
aifc[de]compress	lossy	2.3	2.2

Table 2: Compression rates and speeds

To investigate the effects of lossy coding on speech recognition performance the test portion of the TIMIT database was coded at four bits per sample and the resulting speech was recognised with a state of the art phone recognition system. Both shorten and the g722 ADPCM standard gave negligible additional errors (about 70 more errors over the baseline of 15934 errors), but it was necessary to apply a factor of four scaling to the waveform for use with the g722 ADPCM algorithm. g722 ADPCM without scaling and the telephony quality g721 ADPCM algorithm (designed for 8kHz sampling and operated at 16kHz) both produced significantly more errors (approximately 500 in 15934 errors). Coding this database at four bit per sample results in approximately another factor of two compression over lossless coding.

Decompression and playback of 16 bit, 44.1 kHz stereo audio takes approximately 45% of the available processing power of a 486DX2/66 based machine and 25% of a 60 MHz Pentium. Disk access accounted for 20% of the time on the slower machine. Performing compression to three bits per sample gives another factor of three compression, reducing the disk access time proportionally and providing 20% faster execution with no perceptual degradation (to the authors ears). Thus real time decompression of high quality audio is possible for a wide range of personal computers.

6 Conclusion

This report has described a simple waveform coder designed for use with stored waveform files. The use of a simple linear predictor followed by Huffman coding according to the Laplacian distribution has been found to be appropriate for the examples studied. Various techniques have been adopted to improve the efficiency resulting in real time operation on many platforms. Lossy compression is supported, both to a specified bit rate and to a specified signal to noise ratio. Most simple sample file formats are accepted resulting in a flexible tool for the workstation environment.

References

[1] John Garofolo, Tony Robinson, and Jonathan Fiscus. The development of file formats for very large speech corpora: Sphere and shorten. In *Proc. ICASSP*, volume I, pages 113–116, 1994.

[2] N. S. Jayant and P. Noll. *Digital Coding of Waveforms.* Prentice Hall, Englewood Cliffs, NJ, 1984. ISBN 0-13-211913-7 01.

[3] R. F. Rice and J. R. Plaunt. Adaptive variable-length coding for efficient compression of spacecraft television data. *IEEE Transactions on Communication Technology*, 19(6):889–897, 1971.

[4] Pen-Shu Yeh, Robert Rice, and Warner Miller. On the optimality of code options for a universal noisless coder. JPL Publication 91-2, Jet Propulsion Laboratories, February 1991.

[5] Robert F. Rice. Some practical noiseless coding techniques, Part II, Module PSI14,K+. JPL Publication 91-3, Jet Propulsion Laboratories, November 1991.

Appendix: The shorten man page (version 1.22)

```
SHORTEN(1)                USER COMMANDS                SHORTEN(1)


NAME
     shorten - fast compression for waveform files

SYNOPSIS
     shorten [-hl] [-a #bytes] [-b #samples] [-c  #channels]  [-d
     #bytes]  [-m  #blocks]  [-n  #dB] [-p #order] [-q #bits] [-r
     #bits]  [-t  filetype]  [-v  #version]   [waveform-file
     [shortened-file]]

     shorten -x [-hl] [ -a #bytes] [-d  #bytes]   [shortened-file
     [waveform-file]]

DESCRIPTION
     shorten reduces the size of waveform files (such  as  audio)
     using  Huffman  coding  of prediction residuals and optional
     additional quantisation. In lossless  mode  the  amount  of
     compression  obtained depends on the nature of the waveform.
     Those composing of low frequencies and low  amplitudes  give
     the  best  compression,  which  may be 2:1 or better. Lossy
     compression operates by specifying a minimum acceptable seg-
     mental  signal to noise ratio or a maximum bit rate.  Lossy
     compression operates by zeroing the lower order bits of  the
     waveform, so retaining waveform shape.

     If both file names are specified then these are used as  the
     input and output files.  The first file name can be replaced
     by "-" to read from standard input and likewise  the  second
     filename can be replaced by "-" to write to standard output.
     Under UNIX, if only one file name is  specified,  then  that
     name is used for input and the output file name is generated
     by adding the suffix ".shn" on compression and removing  the
     ".shn"  suffix  on  decompression.  In these cases the input
     file is removed on completion.  The use  of  automatic  file
     name generation is not currently supported under DOS.  If no
     file names are specified, shorten reads from standard  input
     and  writes to standard output.  Whenever possible, the out-
     put file inherits the permissions, owner, group, access  and
     modification times of the input file.

OPTIONS
```

-a align bytes

Specify the number of bytes to be copied verbatim before compression begins. This option can be used to preserve fixed length ASCII headers on waveform files, and may be necessary if the header length is an odd number of bytes.

-b block size

Specify the number of samples to be grouped into a block for processing. Within a block the signal elements are expected to have the same spectral characteristics. The default option works well for a large range of audio files.

-c channels

Specify the number of independent interwoven channels. For two signals, a(t) and b(t) the original data format is assumed to be a(0),b(0),a(1),b(1)...

-d discard bytes

Specify the number of bytes to be discarded before compression or decompression. This may be used to delete header information from a file. Refer to the -a option for storing the header information in the compressed file.

-h Give a short message specifying usage options.

-l Prints the software license specifying the conditions for the distribution and usage of this software.

-m blocks

Specify the number of past blocks to be used to estimate the mean and power of the signal. The value of zero disables this prediction and the mean is assumed to lie in the middle of the range of the relevant data type (i.e. at zero for signed quantities). The default value is non-zero for format versions 2.0 and above.

-n noise level

Specify the minimum acceptable segmental signal to noise ratio in dB. The signal power is taken as the variance of the samples in the current block. The noise power is the quantisation noise incurred by coding the current block assuming that samples are uniformally distributed over the quantisation interval. The bit rate is dynamically changed to maintain the desired signal to noise ratio. The default value represents lossless coding.

-p prediction order

Specify the maximum order of the linear predictive filter. The default value of zero disables the use of linear prediction and a polynomial interpolation method is used instead. The use of the linear predictive filter generally results in a small improvement in compression ratio at the expense of execution time. This is the only option to use a significant amount of floating point processing during compression. Decompression still uses a minimal number of floating point operations.

Decompression time is normally about twice that of the default polynomial interpolation. For version 0 and 1, compression time is linear in the specified maximum order as all lower values are searched for the greatest expected compression (the number of bits required to transmit the prediction residual is monotonically decreasing with prediction order, but transmitting each filter coefficient requires about 7 bits). For version 2 and above, the search is started at zero order and terminated when the last two prediction orders give a larger expected bit rate than the minimum found to date. This is a reasonable strategy for many real world signals - you may revert back to the exhaustive algorithm by setting -v1 to check that this works for your signal type.

-q quantisation level

Specify the number of low order bits in each sample which can be discarded (set to zero). This is useful if these bits carry no information, for example when the signal is corrupted by noise.

-r bit rate

Specify the expected maximum number of bits per sample. The upper bound on the bit rate is achieved by setting the low order bits of the sample to zero, hence maximising the segmental signal to noise ratio.

-t file type

Gives the type of the sound sample file as one of {ulaw,s8,u8,s16,u16,s16x,u16x,s16hl,u16hl,s16lh,u16lh}. ulaw is the natural file type of ulaw encoded files (such as the default sun .au files). All the other types have initial s or u for signed or unsigned data, followed by 8 or 16 as the number of bits per sample. No further extension means the data is in the natural byte order, a trailing x specifies byte swapped data, hl explicitly states the byte order as high byte followed by low byte and lh the converse. The default is s16, meaning signed 16 bit integers in the natural byte order.

Specific optimisations are applied to ulaw files. If lossless compression is specified then a check is made that the whole dynamic range is used (useful for files recorded on a SparcStation with the volume set too high). If lossy compression is specified then the data is internally converted to linear. The lossy option "-r4" has been observed to give little degradation.

-v version

Specify the binary format version number of compressed files. Legal values are 0, 1 and 2, higher numbers generally giving better compression. The current release can write all format versions, although continuation of this support is not guaranteed. Support for decompression of all earlier format versions is guaranteed.

-x extract

Reconstruct the original file. All other command line options except -a and -d are ignored.

METHODOLOGY

shorten works by blocking the signal, making a model of each block in order to remove temporal redundancy, then Huffman coding the quantised prediction residual.

Blocking

The signal is read in a block of about 128 or 256 samples, and converted to integers with expected mean of zero. Sample-wise-interleaved data is converted to separate channels, which are assumed independent.

Decorrelation

Four functions are computed, corresponding to the signal, difference signal, second and third order differences. The one with the lowest variance is coded. The variance is measured by summing absolute values for speed and to avoid overflow.

Compression

It is assumed the signal has the Laplacian probability density function of exp(-abs(x)). There is a computationally efficient way of mapping this density to Huffman codes, The code is in two parts, a run of zeros, a bounding one and a fixed number of bits mantissa. The number of leading zeros gives the offset from zero. Signed numbers are stored by calling the function for unsigned numbers with the sign in the lowest bit. Some examples for a 2 bit mantissa:

```
100  0
101  1
110  2
111  3
0100 4
0111 7
00100     8
0000100   16
```

This Huffman code was first used by Robert Rice, for more details see the technical report CUED/F-INFENG/TR.156 included with the shorten distribution as files tr154.tex and tr154.ps.

SEE ALSO

compress(1),pack(1).

DIAGNOSTICS

Exit status is normally 0. A warning is issued if the file is not properly aligned, i.e. a whole number of records could not be read at the end of the file.

BUGS

There are no known bugs. An easy way to test shorten for your system is to use "make test", if this fails, for whatever reason, please report it.

No check is made for increasing file size, but valid waveform files generally achieve some compression. Even compressing a file of random bytes (which represents the worst case waveform file) only results in a small increase in the file length (about 6% for 8 bit data and 3% for 16 bit data).

There is no provision for different channels containing different data types. Normally, this is not a restriction, but it does mean that if lossy coding is selected for the ulaw type, then all channels use lossy coding.

It would be possible for all options to be channel specific as in the -r option. I could do this if anyone has a really good need for it.

See also the file Change.log and README.dos for what might also be called bugs, past and present.

Please mail me immediately at the address below if you do find a bug.

AVAILABILITY

The latest version can be obtained by anonymous FTP from svr-ftp.eng.cam.ac.uk, in directory comp.speech/sources. The UNIX version is called shorten-?.??.tar.Z and the DOS version is called short???.zip (where ? represents a digit).

AUTHOR