Two-Channel Coding with Application to Physical Mail

Bertrand Haas
Pitney Bowes, MS 26-31
Shelton, CT 06484
USA

Abstract

We consider here the situation where two or more parallel channels of different capacity and different robustness to noise are simultaneously used to communicate a message. We propose a scheme that takes advantage of this situation to combine compression and error handling. We demonstrate the advantage of this scheme through an implementation in physical mail.[1]

Introduction

The preferred situation in Shanon's Theory of Communication is that of a single channel. However, in many real life applications it makes sense to distinguish between two or more channels during communication. For instance, it is often the case that the accuracy of transmission of an image is much higher than the human accuracy of perception. This allows the transmission of subliminal information at the same time than the intended human perceivable image information. Information Hiding (watermarking and steganography) extensively uses the subliminal channel capacity (while lossy data compression tends to reduce it). However data hidden in images is often more sensitive to degradation due to noise. In other words, the subliminal channel is more *fragile*. We consider here two coexisting channels, one fragile and one robust. If the robust channel had much larger capacity than the fragile one, the advantage of using both would fade out. Therefore we assume some capacity constraint on the robust channel relative to the fragile one. This is exactly the situation in the physical postal application described in section "Application to a physical mail system".

For simplicity of exposition, we consider here the transmission of an alphanumeric message (with an alphabet of more than 2 characters) coded as a binary string. The generation of a message is often modeled according to the iid (Independent Identically Distributed random variables) model. It is a convenient model since it is easy to work with, but it is mostly a first approximation, in particular for English text, where correlation between characters is clear (for example "t" and "h" are often adjacent). For clarity of exposition we will develop our compression scheme within the iid model, but it is understood that some additional steps would make it work as well in a more accurate model. For instance, the alphabet can be expanded to include pairs of characters that are highly correlated (like "th").

A long message has to be compressed before being transmitted through any channel. The best compression algorithms usually use binary strings of variable lengths to encode characters. A typical such algorithm is Huffman coding. It is provably the best algorithm in the iid model, but it suffers from "fragility" (like most variable length coding). Indeed if a bit error occurs in the compressed binary string during transmission, the rest of the message is mostly unrecoverable. To avoid this problem, a good error correction algorithm is necessary; with the obvious drawback of size increase. This combination of compression and error correction results in removing useless redundancy and adding useful ones. However, in many applications, error correction is too much of a luxury, as the increased size of the message becomes prohibitive, and softer error handling is sufficient. For instance, electronic packet transmission often requires only error detection; if an error is detected the packet is retransmitted. In some applications a few errors might be tolerable and only error containment is sufficient, that is, a bit error only affects the corresponding codeword and not the rest of the message.

The compression algorithm we propose here takes advantage of the presence of a robust channel of lower capacity and a fragile channel of higher capacity. The output of the compression is a pair of binary strings. A shorter *fragile* (in the same sense as in Huffman coding) *string* that is intended to be sent through the robust channel, and a longer *robust* (in the sense of error containment) *string* that is intended to be sent through the fragile channel. We therefore combine efficient compression and error handling in one step.

The Compression Algorithm

The variable input of the algorithm is a string of characters and the output is a pair (*robust string*, *fragile string*). The parameter input are two dictionaries (which are made public). We assume we have a large sample of messages,

[1]The author thanks Leon Pintsov for his motivating suggestions

in order to gather the statistic parameters necessary to construct these dictionaries.

The first codeword dictionnary

Let m be the size of the character alphabet. A character frequency count on a large sample of *initial messages* is first performed. The characters are then ordered by decreasing frequency. A code dictionary is then constructed by associating binary strings to the characters in the following way: The characters between the positions $2^i - 1$ and $2^{i+1} - 2$ are associated with all the binary strings of length i (up to the length of the alphabet). The order in which the strings are associated to the characters within this range is unimportant for the sole purpose of compression. So the two first (therefore most frequent) characters are coded with the length one strings ”0” and ”1”.

The robust and the raw fragile string

From an initial message (a string of characters), a binary robust string is produced simply by replacing the characters of the messages by the corresponding codewords of the first dictionary. At the same time, a ”raw” fragile string (non binary) is produced by sequentially recording the bit length of the codewords for each character of the message. To decode the pair of strings, one places periods in the robust string at the positions specified by the fragile string. This delimitates the codewords, and one can then replace each codeword by its associated character using the first dictionary. The reason why the two strings are called ”robust” and ”fragile” now becomes clear: If one error occurs in the fragile string all the periods there and after will be shifted, and the rest of the robust string will be wrongly decoded. If one bit error occurs in the robust string, then the error is confined to its codeword and does not affect the rest of the decoding.

The second codeword dictionary and the fragile string

The raw fragile string still has to be encoded to produce the final binary string. Here the characters of the raw fragile string are lengths of codewords of the first dictionary. So if L_1 is the length of the first alphabet (the characters for the initial messages) , the length L_2 of the second alphabet (the characters for the raw fragile strings) is:

$$L_2 = \mathrm{ceil}(\log_2(L_1))$$

that is, substantially smaller than L_1. So the second alphabet can be coded with $\mathrm{ceil}(\log_2(L_2))$ bits per character. However, we can do better by compressing again the raw fragile string. Since the correlation between lengths of codewords in the robust string can be expected to be much lower than the correlation between codewords themselves, the iid model can be expected to be rather good for the generation of raw fragile strings. Huffman coding is therefore a natural choice. Moreover, the raw fragile string being already fragile in the sense described above, encoding it with the Huffman algorithm will not really make it more fragile. The large sample of raw fragile strings is used to construct the Huffman tree and the associated dictionary (we call it the *second dictionary*). Raw fragile strings can then now be Huffman encoded to produce the final fragile strings.

Application to a physical mail system

An *indicium* is a postage label that is printed directly on the mail piece (or perhaps on a sticker to be appended to the mail piece) and that acts as a proof of payment for the postal service. We consider here the generation by a printer-meter of an indicium that contains several parts, among which only two are of interest for our purpose: a variable grey level image of high enough complexity so that a substantial amount of information can reliably be hidden in it; and a two dimensional Datamatrix barcode with some standard information (meter identification number, some meter accounting data, postage denomination, etc.) encoded and cryptographically signed. See figure 1.

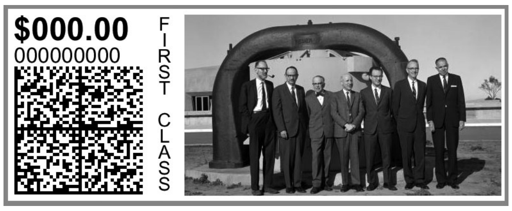

Figure 1: *An example of indicium*

The robust and the fragile channels

The data capacity of a barcode is mostly taken by the standard information and the cryptographic signature, and only 20 bytes are available to embed other kinds of information. Since the Datamatrix barcode is a very simple monochrome graphic designed to be read by a machine after being printed on papers from a wide range of quality, and since it has error correction (Reed-Solomon) built-in, we can consider it a robust channel, with limited capacity (20 bytes) for our purpose.
The fragile channel is the image together with a watermarking algorithm that allows to have a minimum of 30 bytes of information embedded into it. The print and scan process always distorts the image and introduces errors when the hidden information is retrieved. In particular, even though the ink in the printer with which the indicium is printed is of high quality, the paper on which it is printed is not under control. As a result the printed image may suffer from poor ink-paper interaction. We use, however, a watermarking algorithm that encodes each bit of

the message in a block. We thus can assume that in recovering the message, bits may be misread but not missed.

The purpose of two-channel compression for the application

In the printer-meter under consideration the recipient addresses are also printed on the mail pieces (at the same handling time than the indicium, but with a different printhead). The occasion to include also some information about the address (for more thorough verification) is not missed, but the preferred way is usually to hash the address to 20 bytes and include the hash in the barcode. The main drawback is that at verification point the address is OCR-read (Optical Character Recognition) and some errors may occur. The resulting hash is then very different than the hash in the barcode and when the two are compared, the mail piece is marked for further investigation. We propose here to use the two-channel coding described above encode the full address, instead of a hash, in both the barcode and the image. At verification point, the address retrieved by decompression is then compared to the OCR-read one, and only in cases where the two are very different will the mail piece be out-streamed.
In order to fit the address into the allowed 20 bytes of robust channel and 32 bytes of fragile channel, we first transform the address by concatenating the three address lines, removing all white characters and making all alphabetic characters upper case. We call the result the *initial (address) string.*

The compression results on addresses

We used a sample of $3,000$ regular addresses to construct the dictionaries. The results are as follows: The frequency distribution of characters is shown on figure 2. The dictionary inferred from these frequencies is shown on table 1. Robust strings and fragile raw strings are then computed. The distribution of the codeword length is shown in table 2 together with the deduced Huffman dictionary for the fragile strings. For instance the address

```
Bertrand Haas
1234 Fifth Avenue
La Bella Citta, AB 09876
```

produces the robust 128 bits string:
10101100010000011010100011110010001101000000
00010000000001010110110000101100101010100100
1001101000000001100010101001111111001111001
and the 109 bits fragile string:
1000101010100111111100100101111010100001100
0011100110001111000111001100011111001111101
0100100000110111010101010

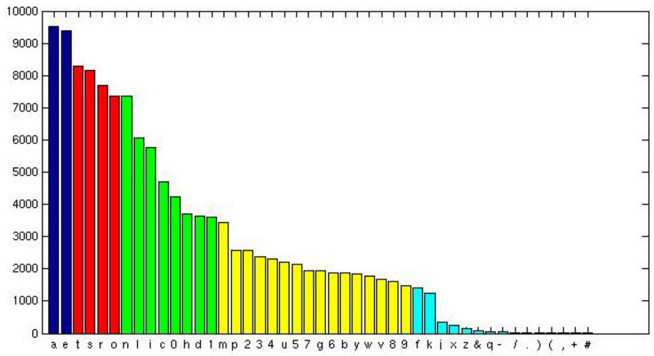

Figure 2: The distribution of characters in the initial address string.

'A'	0	'P'	0001	'F'	00000
'E'	1	'2'	0010	'K'	00001
'T'	00	'3'	0011	'J'	00010
'S'	01	'4'	0100	'X'	00011
'R'	10	'U'	0101	'Z'	00100
'O'	11	'5'	0110	'&'	00101
'N'	000	'7'	0111	'Q'	00110
'L'	001	'G'	1000	'-'	00111
'I'	010	'6'	1001	'/'	01000
'C'	011	'B'	1010	'.'	01001
'0'	100	'Y'	1011	')'	01010
'H'	101	'W'	1100	'('	01011
'D'	110	'V'	1101	','	01100
'1'	111	'8'	1110	'+'	01101
'M'	0000	'9'	1111	'#'	01110

Table 1: The first dictionary

Statistical results

In table 3 we summarized the mean, standard deviation, minimum and maximum, of the bit lengths of the following: The initial address encoded with 8 bits, The robust strings, the fragile strings, and to gauge the compression efficiency, the total length (the sum of the two previous) to be compared with the length of the full Huffman encoded addresses. We collected these parameters on the same sample of $3,000$ addresses we used to construct the dictionaries.

We see that the maximal length for the robust strings, 193 bits, is below the capacity of the watermark ($32 \times 8 =$

'3'	39064	11
'4'	33661	10
'2'	31517	01
'1'	18931	001
'5'	3663	000

Table 2: The second dictionary (with character frequencies)

	mean	std. dev.	min.	max.
initial address	338.1	55.4	160	568
robust string	117.3	19.2	64	193
fragile string	92	15.6	41	158
total length	209.3	34.2	105	347
Huffman encoded	202	32.6	100	344

Table 3: Statistical results on bit lengths

256 bits), and the mean length, 117.3 bits, is less than half this capacity. This means that we can optionally add some redundancy, in the form of error correction coding, to the addresses to make them more robust to the print-scan channel.
The maximum length of the fragile string, 158 bits, is right below the allowed capacity of the barcode ($20 \times 8 = 160$). It may happen that an address produces a fragile string longer than 160 bits even though some user limitations to the length of the address input is embedded in the printer. In that case, it is always possible to crop the initial address string of some characters in order to shorten the fragile string below 161 bits.
The compression rate (length of compressed address divided by length of initial address) averages 61.9% for two-channel coding and 59.8% for Huffman coding. We thus lose 1.1% in compression rate, but we gain in error robustness for 56% of the compressed message, that is a rather good trade-off.

The MATLAB scripts

We include here below two MATLAB scripts used to implement our compression algorithm. Their both input a $3 \times n$ cell array (the three rows correspond to the standard three lines of the addresses, and the n columns to n addresses; n should be large for the first script. The first script produces a structure C with fileds C.alphab (the alphabet of the initial address strings), C.freq (the frequencies of the alphabet characters), and C.cwords (the codewords associated to the alphabet characters). The alphabet and codewords are ordered by decreasing frequencies. The second script encodes addresses and takes also as input the code computed by the first script, and the Huffman dictionary (to be computed with another script). The output is a structure B with fields B.rob (the robust string) and B.frag (the fragile string).

```
function C = makeTCcode(A)
S = strcat(A{:});
S = upper(S);  S = regexprep(S,' ','');
numS = uint8(S);
freq = hist(numS, 32:126);
pos = find(freq);
alphcar = char(pos+31);
alphcell = cellstr(alphcar')';
freq = freq(pos);
[ofreq, ix] = sort(freq,'descend');
C.alphab = alphcell(ix);
C.freq = ofreq;
n = length(C.freq);  C.cwords = cell(1,n);
for i = 1:ceil(log2(n))
   c = cellstr(num2str(dec2bin(0:(2^i- )))));
   c = c(1:min((n - 2^i + 2), 2^i));
   C.cwords((2^i-1):min(n,(2^(i+1)-2)))=c;
end
```

```
function B = dualencode(A1,C,Hf)
A = strcat(A1{:});
A = regexprep(A,' ','');  A = upper(A);
n = length(A);  pos = zeros(1,n);
for i = 1:n
    pos(i) = strmatch(A(i), C.alphab);
end
B.rob = '';
for i = 1:n
    B.rob = strcat(B.rob, C.cwords(pos(i)));
end
B.rob = char(B.rob);
frag = [];
for i = 1:n
   frag = [frag length(C.cwords{pos(i)})];
end
B.hfrag = ...
huffencode(cellstr(num2str(B.frag)),Hf);
```

References

1. Khalid Sayood, Introduction to Data Compression, Morgan Kauffmann (2000).

Biography

Bertrand Haas received his Ph.D. in Mathematics from the University of Basel in Switzerland in 1998. After a postdoctoral year at the Fields Institute and the University of Toronto, a subsequent year at the Mathematical Science Research Institute and the University of California at Berkeley, and two years of professorship at Michigan State University, Bertrand joined the Research and Development team at Pitney Bowes. Since then he has been working on cryptography, coding theory and image processing and their many applications to concrete engineering problems. He is a member of IEEE.