*Range encoding: an algorithm for removing redundancy from a digitised message.

G. N. N. Martin

IBM UK Scientific Center

Presented in March 1979 to the Video & Data Recording Conference, held in Southampton July 24-27 1979.

Redundancy in a message can be thought of as consisting of contextual redundancy and alphabetic redundancy. The first is illustrated by the fact that the letter Q is nearly always followed by the letter U, the second by the fact that the letter E is far more common than the letter X. Range encoding is an algorithm for removing both sorts of redundancy.

Since Huffman [1] published his paper in 1952 there has been a number of papers, e.g. [2], describing techniques for removing alphabetical redundancy, mostly generating prefix codes, and mostly transforming the messages into a bit string. The usual aim of such techniques is to reduce the quantity of storage required to hold a message.

In the last fifteen years the growth of telemetry has increased interest in techniques for removing contextual redundancy. Many of these techniques approximate the message, rather than simply remove redundancy. Such techniques are often analog, and include transmitting the difference between a measured signal and a prediction of that measurement, or varying the rate at which a value is sampled according to the recent measurements of that value [3,4]. The output of such techniques may be a signal of generally low amplitude, or an intermittent signal ; the usual aim being to decrease the power consumed by a transmitter, or to reduce the risk of a circuit or recording medium being overloaded.

Many techniques are almost optimal in a wide variety of situations, but none are universally applicable. In contrast, range encoding may be used to remove all the redundancy that we can describe in any digitised message. It can produce encodings to any base.

Nomenclature

We shall consider an uncoded or decoded message as a string of letters drawn from an alphabet, and an encoded message as a string of digits to a given base, thus it will be obvious whether we are talking about an encoded message or an uncoded one. We shall require the probability of a given letter occuring in any given context to be described by a frequency algorithm.

We shall illustrate our algorithm by encoding and decoding a message composed of letters drawn from the alphabet {K,L,M,N}, and forming an encoded string of digits to base ten.

- Original manuscript sourced from Dr.Glen Langdon, (ex IBM), Nov. 96 at langdon@cse.ucsc.edu
  Computer File typed in using **world's worst** word processor by Kavita Karanth, Nov '97

Range encoding

If we say that a storage medium has a width of s, or a width of d digits of base b, we mean that it can take one of s, or one of b^d, different values. If we do not specify that the width is in digits, then we are using absolute numbers. If we store a letter in the storage medium and so restrict the medium to taking one of t different values, then the width of the encoding of the letter is s/t, and the remaining width is t, in which we can store a remainder of width t. The set of t different values that can represent the letter is the range of the letter in the width of storage. For example, if the range of a letter in a byte of storage of width 256 is $(n|240 \leq n<250)$ then the width of the letter is 25.6, and the remaining width is 10. We can store as remainder anything that we could store in a decimal digit.

We have assumed that we can treat the value of storage as a number: the mapping of the s possible values of storage onto the integers from 0 to s-1 is usually natural. Let us write $(n|B \leq n<T)$ as [B,T). If a range has the form [B,T) ,then we can combine it with a remainder by simple arithmetic. Thus if $i \in [0,T-B)$ is to be stored as remainder to [B,T) then the storage takes the value B+i ; or if $[i , j) \subseteq [0,T-B)$ is stored as partial remainder to [B,T), then the storage is constrained to [B+i,B+j).

Let fa be the probability of the letter 'a' occurring in any given context. We assume our alphabet to be ordered and define Fa to be the probability of a letter preceding 'a' in the alphabet occuring in the same context, thus:

$$Fa = \sum_{x<a} fx$$

Shannon [via 5] showed that to minimise the expected number of digits to base b required to represent a message, we should encode each letter 'a' so that its width is $-\log_b$ (fa) digits, i.e. its absolute width is 1/fa. We can not necessarily manage this exactly, but if we let the encoding of 'a' in storage of width s be $[\lfloor s.Fa \rfloor, \lfloor s(fa + Fa) \rfloor)$ then the width of the letter approaches 1/fa very closely for s.fa>>1. Observe that provided for all 'a' $s.fa \geq 1$, then each letter is encodable and unambiguously decodable.

Note that we write fa and Fa rather than the more conventional f(a) and F(a), and that in future we shall simply write sfa rather than s.fa.

Decoding

A letter 'a' together with its remainder will encode in storage of width s as $i \subseteq [\lfloor aFa \rfloor, \lfloor s(Fa + fa) \rfloor)$. Let L be the last letter e in the alphabet for which Fe<j. We can use L to deduce 'a' given i, for :

$$\lfloor sFa \rfloor \leq i < \lfloor s (Fa + fa) \rfloor$$

$$\therefore \quad sFa< i+1\leq s(Fa+fa)$$

$$\therefore \quad Fa < \frac{i+1}{s} \leq Fa+fa$$

$$\therefore a = L(\frac{i+1}{s})$$

However we must take account of rounding errors in the calculations of $\frac{i+1}{s}$.

We can always verify the letter, and correct it if necessary by confirming the top line above, namely:

$$\lfloor sFa \rfloor \leq i < \lfloor s(Fa + fa) \rfloor$$

Having deduced 'a' the remainder is $i - \lfloor sFa \rfloor$, and was encoded in a width of $\lfloor s(Fa + fa) \rfloor - \lfloor sFa \rfloor$.

A basic algorithm

Let Ai be the i'th letter of a message that we wish to encode, $1 \leq i \leq k$. Imagine we choose some large storage of width s into which to code A1, leaving a remaining width of R1 in which we code A2, leaving a remaining width of R2 in which we code A3, and so on. The widths are given by

$$R(0) = s, \quad Ri = \lfloor R(i-1)(FAi + fAi) \rfloor - \lfloor R(i-1)FAi \rfloor$$

Then if $Bj = \sum_{i=1}^{j} \lfloor R(i-1)\, FAi \rfloor$, the range of the complete message in the storage of width s would be

[Bk,Bk+Rk).

Figure 1 illustrates such an encoding, the message 'NMLNNNKKNML' encoding in storage of width 10^{11} as the range [74360239870,74360281886). If we choose a number in the middle of this range, then we need only store or transmit the leading seven digits, since whatever the trailing four digits are taken to be, the number stays in range. Thus our message encodes as '7436026'.

In fact if an encoding leaves a remaining width of r then at least the trailing $\lfloor \log_b(r/2) \rfloor$ digits are insignificant, b being the base of the encoding (at most the trailing $\lfloor \log_b r \rfloor$ digits are insignificant).

A revised algorithm.

The length of the message that can be encoded using the basic algorithm is limited by the size of integer that the encoder can manipulate. We shall now revise the algorithm to remove this limitation.

If a letter 'a' encodes in storage of width s as [B,T) the remaining width is T-B. If T-B is too small for our purpose, then by adding a trailing digit (base b) of storage the range of storage becomes [Bb,Tb), and the remaining width becomes (T-B)b. Note that when decoding 'a' we must ignore this extra digit, since the encoding of 'a' in storage of width sb is not necessarily [Bb,Tb).

Let $s=b^w$ where w is the largest whole number of digits base b that our encoder can conveniently handle. We shall encode the first letter of a message in storage of width s, and we shall then add as many trailing digits of storage as we may without causing the remaining width to exceed s. Let the storage after encoding the i'th letter be of width Si and

Figure 1 Range encoding in wide storage

This figure illustrates how we could encode a short message in storage of width 10^{11}. The first letter is encoded as a range in the whole storage, then each subsequent letter is encoded in the remaining width of the encoding so far.

The frequency algorithms f and F are represented by the following table.

a	fa	Fa
K	0.1	0
L	0.21	0.1
M	0.27	0.31
N	0.42	0.58

The message to be encoded is ‘NMLNNNKKNML’

Remaining Width	Next letter	Range of next letter	Message so far	Range of message so far
100000000000	N	[58000000000 , 100000000000)	N	[58000000000 , 100000000000)
42000000000	M	[13020000000, 24360000000)	NM	[71020000000, 82360000000)
11340000000	L	[01134000000, 03515400000)	NML	[72154000000, 74535400000)
02381400000	N	[01381212000, 02381400000)	NMLN	[73535212000, 74535400000)
01000188000	N	[00580109040, 01000188000)	NMLNN	[74115321040, 74535400000)
00420078960	N	[00243645796, 00420078960)	NMLNNN	[74358966836, 74535400000)
00176433164	K	[00000000000, 00017643316)	NMLNNNK	[74358966836, 74376610152)
00017643316	K	[00000000000, 00001764331)	NMLNNNKK	[74358966836, 74360731167)
00001764331	N	[00001023311, 00001764331)	NMLNNNKKN	[74359990147, 74360731167)
00000741020	M	[00000229716, 00000429791)	NMLNNNKKNM	[74360219863, 74360419938)
00000200075	L	[00000020007, 00000062023)	NMLNNNKKNML	[74360239870, 74360281886)

The complete code must be quoted to seven significant digits,
e.g. 7436026

of value [Bi, Ti) ; then we shall encode the next letter A(i+1) in storage of width R(i+1) where:

$$R(i+1) = (Ti-Bi)b^{k(i+1)}$$
$$k(i+1) = w-\lceil \log_b (Ti - Bi) \rceil$$

Thus for any i>0

$$[Bi, Ti) = [B(i-1)b^{ki} + \lfloor RiFAi \rfloor, B(i-1)b^{ki} + \lfloor Ri(FAi + fAi) \rfloor)$$

and

$$Si = \sum_{j=1}^{i} kj \text{ digits.}$$

where,
[B0,T0) = [0,1)

An example will make this algorithm more obvious. Figure 2 shows our sample message being encoded with s = 1000.

<u>Implementation</u>

Consider the range [B,T) of storage immediately before a further letter is added in, and let s be the upper bound of T-B. Observe that we can identify three (possibly empty) zones within the digits that compose any number in the range; for example if s = 1000 then [B,T) might be

```
     [1319314 ,
      1320105 ]
      |  |  |    |
 zone   1  2   3
```

Remember that T-1, not T, is the highest number in the range.

Zone 1 consists of digits that are common to every number in the range, and thus are unaffected by the choice of remainder. These digits may be <u>committed</u> to the transmitter or to storage.

Zone 2 consists of n digits forming a number db^{n-1} or $db^{n-1} - 1$, where d is a single digit and b is the base of the encoding. In our example n=2 and d=2. Zone 2 is the digit that may be affected by the choice of remainder, but which are not required in order to distinguish between two numbers in the range. We shall call these the <u>delayed</u> digits, and (d,n) identifies the possible values of the delayed digits. By convention, if n=0 then d=0.

Zone 3 consists of the rightmost w digits, and is sufficient to distinguish between any two numbers from the range.

Consider the range [B',T'), with committed digits c, and delayed digits represented by (d , n). Let x be the committed digits after resolving the delay high, i.e.

$$x = cb^n + db^{n-1}$$

Figure 2 Range encoding in narrow storage

Here we re-encode the message ‘NMLNNNKKNML’ using the same frequency algorithm as in figure 1, but using an encoding algorithm that encodes individual letters in storage of width less than 1000.

Adjusted remainingwidth	Next letter	Range of next letter	Message so far	Range of message so far	Remaining width
1000	N	[580 , 1000)	N	[580 , 1000)	420
420	M	[130 , 243)	NM	[710 , 823)	113
113	L	[011 , 035)	NML	[721 , 745)	24
240	N	[139 , 240)	NMLN	[7349 , “450)	101
101	N	[058 , 101)	NMLNN	[7407 , “450)	43
430	N	[249 , 430)	NMLNNN	[74319 , “”500)	181
181	K	[000 , 018)	NMLNNNK	[74319 , “”337)	18
180	K	[000 , 018)	NMLNNNKK	[743190 , “””208)	18
180	N	[104 , 180)	NMLNNNKKN	[7432004 , “”””080)	76
760	M	[235 , 440)	NMLNNNKKNM	[74320275 , “”””””480)	205
205	L	[020 , 063)	NMLNNNKKNML	[74320295 , “”””””338)	43

The complete code must be quoted to seven significant digits,
e.g. 7432031.

---

then we shall express [B’,T’] as

c,(d,n),[B,T]

where B=B’-xs, and T=T’-xs. For example, [1319314,1320105) becomes 13,(2,2),[-686,105).
The remaining width is T-B and if we combine c,(d,n),[B,T] with the partial remainder $[i,j) \subseteq [0,T\text{-}B]$ then we create the range c,(d,n),[B+i, B+j].
If $B+j \leq 0$ then we may resolve the delay low: if $B+i \geq 0$ then we may resolve the delay high.
Figure 3 shows all the interesting possibilities that can arise.

<u>Figure 3</u> <u>Illustrating c,(d,n),[B,T] + [i,j]</u>

This shows the effect of encoding a letter as partial remainder to the range 13,(2,2),[-686,105], and adjusting the resulting range so that the remaining width is as high as possible without exceeding 1000.

Case 1. The letter encodes in storage width 791 as [000,080)

13,(2,2),[-686,105) + [000,080) ⇒13,(2,2),[-686,-606)
⇒ 1319,(0,0), [314,394) ⇒ 13193, (0,0), [140,940)

Case 2. The letter encodes in storage width 791 as [620,700)

13,(2,2), [-686,105) + [620,700) ⇒ 13,(2,2), [-066,014)
⇒ 13,(2,3), [-660,140)

Case 3. The letter encode in storage width 791 as [700,791)

13,(2,2), [-686,105) + [700,791) ⇒ 13,(2,2),[014,105)
⇒ 1320,(0,0) , [014,105) ⇒ 1320,(1,1), [-860,050)

We have now reduced the ranges to a form that we can implement easily
since if the range is c,(d,n),[B,T) then :

$-s<B<T\leq +s$

d is a single digit

n is a small integer

c need not be held in the encoder/decoder.

We have one further refinement before our algorithm is complete. It is most unlikely that the number of delayed digits will ever grow very large, but we may wish to impose an upper limit, One way in which we may force resolution of the delay is to reduce the top of the range, or to increase the bottom of the range. Thus, for example,

13,(2,3),[-660,140) ⇒ 13,(2,3), [-660,000) ⇒ 13199,(0,0), [340,1000)

or

13,(2,3), [-140,660) ⇒ 13,(2,3),[000,660) ⇒ 13200,(0,0), [000,660)

This wastes at most one bit of storage.

Observations

Sort order : The sort order of encoded messages is the same as the sort order implied for uncoded messages by the alphabetic order chosen in the implementation of the frequency algorithms. In [2] this is called the strong alphabetic property.

Prefix codes : Prefix encoding (e.g. Huffman encoding) is the most popular encoding for removing alphabetic redundancy, so it is pleasing to find that any prefix encoding can be generated or read using the range encoding algorithm that we have developed.

Consider a message encoded using a prefix encoding, where any letter 'a' encodes to a string of digits of length ua and numerical value va. The same message will encode to the same encoding using the range encoding algorithm if we define $Fa=b^{-ua}va$ and $fa=b^{-ua}$ for all 'a', where b is the base of both encodings.

The corollary is that any messages encoded in a single context will form an encoding that can be treated as a prefix encoding if for all 'a', fa is a power of b and Fa/fa is an integer.

Recognising end of message : The decoder is driven by whatever wants the message, and it is the responsibility of the driver to recognise the end of a message. If the driver continues to ask for letters after the end of a message, it will get spurious letters. If the message is not self delimiting we must add a letter 'end -of-message' to the alphabet.

Context

Since f and F map letters in context to probabilities, we should properly talk about fca, Fca, and Lca, where fca is the probability of encountering the letter 'a' in context c, and similarly for F and L. In our example up till now there has been only one context; we shall now derive F and L for an example involving several contexts.

In 1952 Oliver modelled [5] a typical television signal as drawn from an alphabet of m levels, where each letter had probability pk^n of differing from the previous letter by n levels in either direction, where $k<1$, and p is a function of the previous letter.

Each level is encoded in the context of the preceeding level, and it can be shown that :

$$Fca = \frac{x+1-k^{a-c}}{x+y} \quad \text{if } c \le a$$

$$\frac{k-k^{a+1}}{x+y} \quad \text{if } c>a$$

$$\text{where } x=k-k^{c+1} \quad \text{and} \quad y=1-k^{m-c}$$

This can easily be implemented if the encoder holds a list of the values of k^i for $0 \le i \le m$.

Lcj is the highest letter 'a' for which Fca<j, i.e. the highest such that :

$$k^{a-c} > 1+x(1-j)-yj \qquad \text{if } c \le a$$
$$k^{a+1} > k-(x+y)j \qquad \text{if } c > a$$

Thus L too can easily be implemented given a list of the values of k^i for $0 \le i \le m$.

<u>The context of improbable letters</u>

s reflects the largest integer that our encoder is built to handle, and until now we have assumed that frequency algorithm f can only be used with an encoder parameterised by s if for all contexts c and letters 'a', $s/b \ge 1/fca$, or fca=0. By fca = 0 we mean that letter 'a' is truely impossible in context c. We shall now consider how we can simply transform any f, F and L so that they meet this constraint.

Consider a context x where r is the width in which we must encode the next letter. The range of the letter is $[\lfloor rFxa \rfloor, \lfloor r(Fxa+fxa) \rfloor)$. If this range is null, i.e. $\lfloor rFxa \rfloor = \lfloor r(Fxa + fxa) \rfloor$, then we cannot encode the letter 'a'. When we encounter such a range, then we will steal one value from the next non-null range above, namely $\lfloor rFxa \rfloor$, to represent the <u>context</u> <u>marker</u> Cy, which marks the fact that the next letter is coded in the context y. The range of Cy is $[\lfloor rFxa \rfloor, \lceil r(Fxa + fxa) \rceil)$.

Now all letters e such that $\lfloor rFxe \rfloor = \lfloor rFxa \rfloor$ will result in the generation of Cy, except perhaps the highest such letter. We shall identify the range of letters that do as $[\alpha,\beta)$ where α is the lowest such letter, and β is next letter above the highest such letter.

Let us consider a letter 'a' for which the range is not null, i.e.$\lfloor rFxa \rfloor < \lfloor r(Fxa + fxa) \rfloor$. If the next possible letter below 'a' causes the generation of any context marker Cz, then the range of 'a' is reduced to $[\lceil rFxa \rceil, \lfloor r(Fxa + fxa) \rfloor)$, since the value $\lfloor rFxa \rfloor$ is stolen to represent Cz. If this reduced range is null, i.e.$\lceil rFxa \rceil = \lfloor r(Fxa + fxa) \rfloor$, then letter 'a' must also generate context marker Cz.

Thus the range of letters $[\alpha,\beta)$ that generate the context marker Cy is all those letters whose range is included in the range of Cy.

$$\gamma \varepsilon [\alpha,\beta) \iff [\lfloor rFx\gamma \rfloor, \lfloor r(Fx\gamma + fx\gamma) \rfloor) \subseteq [\lfloor rFx\alpha \rfloor, \lceil r(Fx\alpha + fx\alpha) \rceil)$$

The context y is a context of improbable letters in which we encode the letter that caused the generation of the context marker Cy.

F and f are defined in the context y by:

$a \in [\alpha,\beta] \quad \rightarrow \quad Fya = (Fxa - Fx\alpha)/ (Fx\beta - Fx\alpha)$

$$fya = fxa /(Fx\beta - Fx\alpha)$$

If we can calculate Fxa - Fxe directly as a floating point number, where 'a' and e are any two letters, then we do not have to work in double precision even when encoding improbable letters. This process may be repeated to any depth, and thus we may (for example) perform any encoding on an eight bit micro processor.

Note that the algorithm still generates prefix codes if for all 'a', fa is a power of the base, and Fa/fa is an integer.

Conclusion

We are now able to separate the task of describing redundancy from the task of removing it. If we can describe it concisely, we can remove it cheaply.

For the sake of brevity, we merely state that messages encoded using range encoding will have an average length little more than $0.5\log_b(2b)$ digits longer than the theoretical optimum. This paper will also be published as a University of Warwick Theory of Computation report, where we shall justify that statement , and include an APL model of a range encoder and decoder.

Acknowledgements

I am greatful to my employers, IBM, for a valuable education award that has enabled me to attend Warwick University to write up this and other ideas. I am particularly gratetful to my supervisor Dr. M. S. Paterson, for his help in the preparation and presentation of this algorithm.

Post script

Since writing this report, two papers by J. J. Rissanen have been brought to my notice [6,7]. The ideas in those papers and in this appear to be closely related, and it will be interesting to compare them in detail.

References

[1] A method for the construction of minimum redundancy codes.
Huffman D. A. , Proc. I.R.E, vol. 4D, no. 9 1952, pp 1098-1101

[2] Variable length binary encodings.
Gilbert E. N. , and Morre E. F., Bell system tech. Journal,
vol 38, 1959, pp 933-967.

[3] Adaptive data compression
Andrews C. A., Davies J. M., and Schwartz G. R., Proc. of IEEE, vol. 55 no 3, March 1967, pp 267-277.

[4] Data compression and adaptive telemetery.
Kortman C. M., Lockheed Missiles and Space Co., Sunnywale, California.

[5] Efficient coding
Oliver B. M., Bell system tech. Journal , July 1952, pp 724-750

[6] Generalised Kraft Inequality and Arithmetic Coding
Rissanen J. J. , IBM journal of R&D, May 1976, pp 198-203

[7] Arithmetic coding
Rissanen J .J., IBM research report no. RJ2174, 30 Jan 1978

In case you are aware of other interesting ancient manuscripts of Arithmetic Coding , please e-mail to Mahesh Naik at mahesh@aftek.com
*

* This effort was funded by a generous grant from <Put your name here>