4.1 Phased-In Codes

Many of the prefix codes described here were developed for the compression of specific types of data. These codes normally contain codewords of various lengths and they are suitable for the compression of data where individual symbols have widely different probabilities. Data where symbols have equal probabilities cannot be compressed by VLCs and is normally assigned fixed-length codes. The codes of this section (also called phased-in binary codes, see Appendix A-2 in [Bell et al. 90]) constitute a compromise. A set of phased-in codes consists of codewords of two lengths and may contribute something (although not much) to the compression of data where symbols have roughly equal probabilities.

Given n data symbols, where $n = 2^m$ (implying that $m = \log_2 n$), we can assign them m-bit codewords. However, if $2^{m-1} < n < 2^m$, then $\log_2 n$ is not an integer. If we assign fixed-length codes to the symbols, each codeword would be $\lceil \log_2 n \rceil$ bits long, but not all the codewords would be used. The case $n = 1{,}000$ is a good example. In this case, each fixed-length codeword is $\lceil \log_2 1{,}000 \rceil = 10$ bits long, but only 1,000 out of the 1,024 possibe codewords are used.

In the approach described here, we try to assign two sets of codes to the n symbols, where the codewords of one set are $m-1$ bits long and may have several prefixes and the codewords of the other set are m bits long and have different prefixes. The average length of such a code is between $m-1$ and m bits and is shorter when there are more short codewords.

A little research and experimentation leads to the following method of constructing the two sets of codes. Given a set of n data symbols (or simply the integers 0 through $n-1$) where $2^m \le n < 2^{m+1}$ for some integer m, we denote $n = 2^m + p$ where $0 \le p < 2^m - 1$ and also define $P \overset{\text{def}}{=} 2^m - p$. We construct $2p$ long (i.e., m-bit) codewords and P short, $(m-1)$-bit codewords. The total number of codewords is always $2p + P = 2p + 2^m - p = (n - 2^m) + 2^m = n$ and there is an even number of long codewords. The first P integers 0 through $P-1$ receive the short codewords and the remaining $2p$ integers P through $n-1$ are assigned the long codewords. The short codewords are the integers 0 through $P-1$, each encoded in $m-1$ bits. The long codewords consist of p pairs, where each pair starts with the m-bit value P, $P+1, \ldots,$ $P+p-1$, followed by an extra bit, 0 or 1, to distinguish between the two codewords of a pair.

The following tables illustrate both the method and its compression performance. Table 4.1 lists the values of m, p, and P for $n = 7$, 8, 9, 15, 16, and 17. Table 4.2 lists the actual codewords.

Table 4.2 is also the key to estimating the compression ratio of these codes. The table shows that when n approaches a power of 2 (such as $n = 7$ and $n = 15$), there are few short codewords and many long codewords, indicating low efficiency. When n is a power of 2, all the codewords are long and the method does not produce any compression (the compression ratio is 1). When n is slightly greater than a power of 2 (such as $n = 9$ and 17), there are many short codewords and therefore better compression.

n	m	$p = n - 2^m$	$P = 2^m - p$
7	2	3	1
8	3	0	8
9	3	1	7
15	3	7	1
16	4	0	16
17	4	1	15

Table 4.1: Parameters of Phased-In Codes.

i	$n = 7$	8	9	15	16	17
0	00	000	000	000	0000	0000
1	010	001	001	0010	0001	0001
2	011	010	010	0011	0010	0010
3	100	011	011	0100	0011	0011
4	101	100	100	0101	0110	0100
5	110	101	101	0110	0101	0101
6	111	110	110	0111	0110	0110
7		111	1110	1000	0111	0111
8			1111	1001	1000	1000
9				1010	1001	1001
10				1011	1010	1010
11				1100	1011	1011
12				1101	1100	1100
13				1110	1101	1101
14				1111	1110	1110
15					1111	11110
16						11111

Table 4.2: Six Phased-In Codes.

The total size of the n phased-in codewords for a given n is

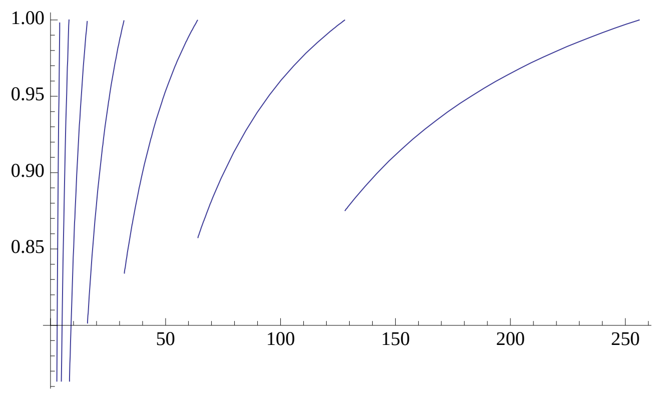
$$P \times m + 2p(m+1) = (2^{m+1} - n)m + 2(n - 2^m)(m+1)$$

where $m = \lfloor \log_2 n \rfloor$, whereas the ideal total size of n fixed-length codewords (keeping in mind the fact that $\log_2 n$ is normally a noninteger), is $n\lceil \log_2 n \rceil$. Thus, the ratio of these two quantities is the compression ratio of the phased-in codes. It is illustrated in Figure 4.3 for n values 2 through 256.

It is obvious that the compression ratio goes up toward 1 (indicating worse compression) as n reaches a power of 2, then goes down (indicating better compression) as we pass that value.

See `http://f-cpu.seul.org/whygee/phasing-in_codes/phasein.html` for other examples and a different way to measure the efficiency of these codes.

Another advantage of phased-in codes is the ease of encoding and decoding them.



```
m:=Floor[Log[2,n]];
Plot[(n m+2 n-2^(m+1))/(n Ceiling[ Log[2,n]]), {n,2,256}]
```

Figure 4.3: Efficiency of Phased-In Codes.

Once the encoder is given the value of n, it computes m, p, and P. Given an integer i to be encoded, if it is less than P, the encoder constructs an m-bit codeword with the value i. Otherwise, the encoder constructs an $(m+1)$-bit codeword where the first m bits (the prefix) have the value $P+(i-P)\div 2$ and the rightmost bit is $(i-P) \bmod 2$. Using $n=9$ as an example, we compute $m=3$, $p=1$, and $P=7$. If $i=6$, then $i<P$, so the 3-bit codeword 110 is prepared. If $i=8$, then the prefix is the three bits $7+(8-7)\div 2=7=111_2$ and the rightmost bit is $(8-7) \bmod 2=1$.

The decoder also starts with the given value of n and computes m, p, and P. It inputs the next m bits into i. If this value is less than P, then this is the entire codeword and it is returned by the decoder as the result. Otherwise, the decoder inputs the next bit b and appends it to $2(i-P)+P$. Using $n=9$ as an example, if the decoder inputs the three bits 111 (equals P) into i, it inputs the next bit into b and appends it to $2(7-7)+7=111_2$, resulting in either 1110 or 1111, depending on b.

The phased-in codes are closely related to the minimal binary code of Section .

See [seul.org 06] for a *Mathematica* notebook to construct phased-in codes.

It is also possible to construct suffix phased-in codes. We start with a set of fixed-length codes and convert it to two sets of codewords by removing the leftmost bit of some codewords. This bit is removed if it is a 0 and if its removal does not create any ambiguity. Table 4.4 (where the removed bits are in italics) illustrates an example for the first 24 nonnegative integers. The fixed-sized representation of these integers requires five bits, but each of the eight integers 8 through 15 can be represented by only four bits because 5-bit codes can represent 32 symbols and we have only 24 symbols. A simple check verifies that, for example, coding the integer 8 as 1000 instead of 01000 does not introduce any ambiguity, because none of the other 23 codes ends with 1000. One-third of the codewords in this example are one bit shorter, but if we consider only the 17 integers from 0 to 16, about half will require four bits instead of five. The efficiency of this code depends on where n (the number of symbols) is located in the

00000	00001	00010	00011	00100	00101	00110	00111
*0*1000	*0*1001	*0*1010	*0*1011	*0*1100	*0*1101	*0*1110	*0*1111
10000	10001	10010	10011	10100	10101	10110	10111

Table 4.4: Suffix Phased-In Codes.

interval $[2^m, 2^{m+1} - 1)$.

The suffix phased-in codes are suffix codes (if c has been selected as a codeword, no other codeword will end with c). Suffix codes can be considered the complements of prefix codes and are also mentioned in Section .