requires 743 bits. In general, if the worst-case number of bits in the coded output must be bounded, a Rice code should be preferred; if the average (assuming m random values) length of the coded sequence is to be minimized, a Golomb code should be used.

Rice codes also have one other significant property compared to Golomb codes: the space of possible parameter values is considerably smaller. If a tabulation technique is being used to determine the parameter for each symbol in the message on the fly, Rice codes are the method of choice.

Generalizations of Elias and Golomb codes have also been described, and used successfully in situations in which geometrically-growing buckets are required, but with a first bucket containing more than one item. For example, Teuhola [1978] describes a method for compressing full-text indexes that is controlled by the vector

$$(b, 2b, 4b, \ldots, 2^k b, \ldots)\,.$$

Jakobsson [1978] and Moffat and Zobel [1992] have also suggested different schemes for breaking values into selectors and buckets.

Another interesting generalization is the application of Golomb and Rice codes to doubly-infinite source alphabets. For example, suppose the source alphabet is given by $S = [\ldots, -2, -1, 0, +1, +2, \ldots]$ and that the probability distribution is symmetric, with $p_{-x} = p_x$, and $p_x \geq p_y$ when $0 \leq x < y$. The standard technique to deal with this situation is to map the alphabet onto a new alphabet $S' = [1, 2, \ldots]$ via the conversion function $f()$:

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ 2x - 1 & \text{if } x > 0 \\ -2x & \text{if } x < 0\,. \end{cases}$$

The modified alphabet S' can then be handled by any of the static codes described in this chapter, with Rice and Golomb codes being particularly appropriate in many situations. But the symmetry inherent in the original probability distribution is no longer handled properly. For example, a Rice code with $k = 1$ assigns the codewords "00", "01", "100", "101", "1100", and "1101", to symbols 0, +1, −1, +2, and −2, respectively, and is biased in favor of the positive values. To avoid this difficulty, a code based upon an explicit selector vector can be used, with an initial bucket containing an odd number of codewords, and then subsequent buckets each containing an even number of codewords. The Elias codes already have this structure, but might place too much emphasis on $x = 0$ for some applications.