Chapter 5

Run Length Coding and the Golomb Code

Introduction

A run-length code is a prefix code that encodes a distribution of nonnegative integers whose distribution resembles the geometric distribution. A run length is defined as the number of successive occurrences of a selected symbol, or a statistical outcome chosen to be the run event. For an alphabet of M-ary outcomes, one (or more) of the outcomes can be chosen as the run event set and the remaining outcomes would belong to the stop event set.

The geometric distribution is based on the binary distribution whose two-valued outcome is either Success or Failure. Section 5.1 characterizes the mathematical form and parameters of the geometric distribution. The name of the geometric distribution is based on its relationship to the geometric series (Equation 5.9). Section 5.5 discusses a change in variable that relates run length values to the random variable of the geometric distribution. Section 5.6 describes and discusses the Golomb code that is the optimal coding for encoding events drawn from a geometric distribution.

5.1 Some Definitions

Both the geometric distribution and the run length distribution for which we employ the Golomb code are derived, in slightly different ways from a binary probability distribution.

Definition 1 (Bernoulli Trial) Two random events, Success (denoted S) and Failure (denoted F), are the only possible outcomes of a Bernoulli trial, and form the binary distribution upon which the random variable of the geometric distribution is based.

The geometric distribution takes on random values 1, 2, ... ∞. In contrast, the Run Length distribution takes on random values 0, 1, 2, ... ∞.

Definition 2 (Run Length) Two random events, Run Continues (denoted C) and Run Stops (denoted S) form the binary distribution upon which the Golomb code for encoding run lengths is based.

Definition 3 (Median) The median value of a random variable is that value such that approximately 50% of the sample point values are above the median, and approximately 50% of the sample point values are below the median.

Definition 4 (Random variable R) The run length random variable R is the number of continue or C outcomes before the stop outcome S occurs. We underline before because the random variable of the geometric distribution includes the final outcome that terminates the sequence of Bernoulli trials.

Observation 1 (Similarity) We use outcome notation S to denote the outcome of the Bernoulli trial for both the random variable families X_G and X_R that terminate the Bernoulli experiment. Outcome S is “Success” to the Bernoulli trial for the geometric distribution, and outcome S is “Stop” for the run length.

Observation 2 (Differences) The difference between the geometric and the run length distribution is a simple matter of what gets counted.

- The geometric distribution counts the total number of samples drawn, including the single success. the total number of Bernoulli trials involved. In the geometric distribution, if the symbol S occurs on the first event, the random variable is 1.
- In run length coding, the random variable R is the number of runs only. If the stop symbol S occurs on the first event, the Run Length is 0, because there were no “run continues” events. This terminology is logical, since how can you have a “run” without the run symbol having occurred?

The Golomb code is a family of codes based on an integer parameter, denoted md, for median. (In [Gol66],...
the notation is \( m \). In the Golomb code, the median run, given \( p(S) \) as the single parameter, is very important. Based upon Defn 3, we expect the following approximation to be true for the median run length.

\[
p^{md}_S \approx \frac{1}{2} \tag{5.1}
\]

Table 5.1 provides, for ranges of \( p_S \), the appropriate value of Golomb parameter \( md \), based on Equation 5.16. In other words, parameter \( md \) is the median value of the run length distribution; i.e., approximately half (50\%) of the occurrence of runs are larger than \( md \), which also means about 50\% of the runs are equal to or shorter than \( md \).

**Question 1** Suppose we knew a runlength \( r \) that we had to code was less than \( md \). How many code bits would be best to tell the decoder the run was less than \( md \)?

**Observation 3** If we could provide this information to the decoder, the decoder has now narrowed the possible values of \( r \) to 0, 1, …, \( md – 1 \).

**Observation 4** (Random variables \( R \) and \( X_G \)) Because of the difference in random variable values between run length coding r.v. \( R \) and the geometric distribution r.v. \( X_G \), the mean \( \mu_G \) of the geometric distribution is \( 1 + \mu_R \), where \( \mu_R \) is the mean value of the run length distribution.

Run length codes are applicable to text files having the property that one or more symbols frequently succeed themselves. Part of the problem of applying run length coding to data compression is to identify good run symbols.

In the case of the pair of binary symbols 0 and 1, the more popular symbol (MPS) of the pair is the run continues symbol \( C \), and the less popular symbol (LPS) as the runs stops symbol \( S \).

Compression applications can have a predetermined criteria for entering run mode. For example, the DAFC algorithm [LR83] compresses bytes in the normal way unless the context has the previous symbol occurring twice in succession. At this time, the DAFC compression algorithm enters run mode and predicts that the next symbol to occur is the previous symbol. The outcome is binary, either the same or not the same as the predicted symbol. Experiments on text files have shown that even if the next symbol is the predicted symbol only 30 to 40 percent of the time, DAFC still enjoys a net gain in compression.

In IBM’s Systems Network Architecture (SNA) for communications, a very simple compression algorithm was devised that identifies repetitive symbols and applies run length coding for compression. Tape drive systems used to archive or back up computer data may also employ a run length code for compressing repeated symbols [OTD90].

Given a run length compression application, a popular code used for run length encodings is called the Golomb code [Gol66]. This code is actually a parameterized family of codes. The code has been proven an optimal prefix code for the geometric distribution.

Before describing the Golomb code, which is based on the geometric distribution, we discuss Bernoulli trials and experiments.

### 5.2 Bernoulli trials and experiments

**Remark 1** The geometric distribution was named after the fact that the values of the countably infinite series formed by the probabilities \( p(x) \) assigned to the series of positive integers \( x \); i.e., \( p(1), p(2), p(3), \text{etc.} \), are the terms in the famous geometric series for parameter \( (p_S) \).

**Definition 5** (Geometric progression) A geometric progression is a sequence of numbers whose successive terms differ by a constant multiplier (e.g., 2): 1, 2, 4, 8, 16, …, \( 2^k \), \ldots.

**Definition 6** (Geometric series) A geometric series is a series whose terms form a geometric progression, such as:

\[
1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} \ldots \tag{5.2}
\]

The geometric series is a particular Bernoulli experiment, and then see how the Golomb code for run length coding is optimal for the geometric distribution. The geometric distribution is a parameterized family of distributions for the positive integers.

**Definition 7** (Bernoulli trial) A Bernoulli trial, as defined in mathematical texts, is a single binary event that either results in success or else in failure.

**Definition 8** (Bernoulli experiment) An experiment, depending on the type, is a fixed or variable number of Bernoulli trials.

The binomial distribution and the geometric distribution are both defined in terms of a Bernoulli experiment. Following a description of the binomial distribution, we describe the geometric distribution.

### 5.3 Bernoulli (or binomial) distribution

This section is a brief diversion from run length coding. We cover this material to illustrate the notion of a Bernoulli trial and its use in the popular Bernoulli distribution. Then we return to its “cousin” probability distribution of interest to this chapter: the geometric distribution.
Definition 9 (Binomial Distribution) A binomial distribution is comprised of a fixed number $N$ of Bernoulli trials. The outcome of the experiment is the number of success or $S$ outcomes from the $N$ samples taken from the binary distribution for the Bernoulli trial: Success or Failure, or from the set $\{S, F\}$.

The outcome of a statistical experiment of $N$ Bernoulli trials is a random variable $X_B$. The probability of success, $p_S$, and the number of trials $N$, uniquely characterize the binomial distribution. From these two values, we can calculate the probability distribution.

The value of the random variable $X_B$ is the number of trials, $1, ..., k, ..., N$ of successes for the experiment. The probability distribution for the experiment of $N$ trials assigns a probability to each possible value that random variable $X_B$ can take on. What might the values of $X_B$ be? Can we have zero successes in $N$ trials? Yes, we certainly can’t have any less than zero, and if all trials end in failure then we have zero successes. Outcome zero has the probability that $N$ failure outcomes occur: $p(F)^N$.

Observation 5 (Independence) Since the trials are independent, we can simply calculate the probability of the sequence of $N$ Bernoulli trials as the product of $N$ probabilities. There are no conditional probabilities to worry about.

Suppose we have $N$ successes out of $N$ trials. So the binomial probability distribution has the following values: \{p(0), p(1), ..., p(N)\}. So we can have 0 through $N$ successes: one more outcome than trials, since 0 successes are possible.

There is only one such way to have $N$ successes, and its probability is:

$$p(S)^N$$

What about outcome 1? The outcome of 1 occurs if only one of the $N$ positions has outcome $S$ and the other $N-1$ positions have outcome $F$.

If the 1 occurs in position 1 and 0 in the others, the probability is:

$$p(S) \times p(F)^{N-1} \quad (5.3)$$

However, the $S$ can occur in any one of the $N$ positions, so we need to multiply Eq 5.3 by $N$ for the probability the outcome of 1 for the binomial distribution. For more complicated number of successes, we need to use the number of combinations for which $s$ successes occur out of $N$ trials. Combinations are covered in Chapter One.

The reason for the name binomial distribution is because the binomial coefficients are used in the description of the binomial distribution. The Bernoulli trials \{1, ..., k, ..., N\} are statistically independent, therefore, the probability $p_S(k)$ of the outcome for the $k^{th}$ trial remains constant independent of the value of $k$. The outcome probabilities of the experiment of $N$ trials can be calculated based on the value of the probability $p_S$ of success.

The probability $p(x)$ of any unique arrangement of $x$ successes and $N-x$ failures is, due to independence, the product of the probabilities, $x$ of which have value $p_S$ (since Bernoulli viewed a success as the less likely outcome) and $N-x$ failures, each failure of probability $1-p_S$:

$$p(x) = p_S^x \times (p_F)^{N-x} \quad (5.4)$$

We need to calculate the value of the probability that random variable of value $x$, a possible value member of the binomial distribution $X_B^N$ takes on, where $x$ denotes the number of successful outcomes out of $N$ trials. To calculate the probability value assigned to $x$, denoted $P(X_B^N = x)$, we need to know how many combinations (arrangements), denoted $Comb(x, N)$, yield $x$ successes and $N-x$ failures. The number of arrangements, $Comb(x, N)$, called the “combination of $x$ things taken $N$ at $x$ time”, is defined by Eqn 5.5 as follows:

$$C_N^x = Comb(x, N) = \frac{N!}{x!(N-x)!} \quad (5.5)$$

Using Eqn 5.5, the number of arrangements for $x$ successes is known, and the calculation for the probability of the number of successes (random variable) $x$ for the binomial distribution is:

$$P(X_B^N = x) = \frac{N!}{x!(N-x)!} p_S^x \times (p_F)^{N-x} \quad (5.6)$$

According to the Law of Large Numbers, the average number of successes in $N$ Bernoulli trials tends to parameter $p_S$ as $N$ tends to infinity.

The multinomial distribution is a generalization of the binomial distribution from two symbols (Success and Failure) to $M$ symbols. In $N$ trials, how many $N_x$ of each symbol $x$ occur?

Example 1 Suppose there are 3 symbols of probabilities $p(0)$, $p(1)$, and $p(2)$, and $N$ is 7. What is the probability that two 0s, three 1s and two 2's occur? This is a joint event and the probability of a particular arrangement (eg., 0 0 1 1 1 2 2) is $p(0)^2 \times p(1)^3 \times p(2)^2$. However, there are $\frac{7!}{2!3!2!}$ such arrangements; so the answer is:

$$\frac{7!}{2!3!2!} p(0)^2 p(1)^3 p(2)^2 \quad (5.7)$$
5.4 Geometric distribution

The geometric distribution, based on random variable $X_G$, has a different experiment based on Bernoulli trials: the number of Bernoulli trials to the first success. Thus, random variable $X_G$ can take on values from the set $\{1, 2, \ldots, \infty\}$, according to whether the success outcome $S$ (experiment stops) on trial number $1, 2, \ldots, \infty$. The value $x$ of random variable $X_G$ takes on values 1, 2, etc., to an arbitrarily large integer. In contrast, the experiment for the Binomial distribution always generates a string of $N$ binary outcomes, and so ends on the $N^{th}$ trial.

**Observation 6** To have an interesting geometric distribution, the probability of “failure” in a Bernoulli trial needs to be at least 0.5, and preferably greater than 0.5. Otherwise, the typical experiment is uninteresting because it terminates after only 1 or 2 trials.

5.4.1 The mean number of Bernoulli trials for Geometric Distribution

We now discuss an intuitive approach to the calculation of the mean $\mu_G$ of the geometric distribution, i.e., of the expected outcome $E(X_G)$. Value $\mu_G$ is the average number of trials per experiment. If the mean is 10, then on average, success occurs with the 10-th trial.

The calculation of mean $\mu_G$ is relatively simple at the intuitive level with a few examples. We know that each trial ends with the outcome whose probability is $p_S$, which in turn is 0.5 or less. So the mean must be greater than one.\(^1\)

The mean $\mu_G$ is always greater than or equal to one (since there is always one success that ends the experiment). On the other hand, there is an average number of $C$ (Bernoulli trial is a failure) so the experiment continues.

**Question 2 (Expectation) So: how many failures outcomes do we expect for each outcome of probability $p_S$).**

Here is an intuitive explanation. Suppose the value of $p_S$ is 1/4. We expect one quarter (1/4) of the trials (symbols $C$ generated) to end an experiment, so there should be $3C$ (continue) outcomes for each experiment.

Think about this: to have a probability distribution, the symbol probabilities sum to one. Since 25% of the time the sequence of Bernoulli trials comprising the experiment must end, then 75% of the time the run continues.

**Observation 7** The experiment includes the outcome that stops the sequence. Over a large number of experiments, the experiments must converge to a mean number of 4 trials per experiment.

**Example 2 (Slicing the II)** Suppose, for example, the average number of failures per success is 2.1416.

By adding 1 for the trial that ended the experiment (a success), the average number of trials per experiment (one success per experiment) must be 3.1416. So the probability failure per trial must average 1/3.1416. Therefore, $\mu_G = \frac{1}{p_S}$.

Now, if you believe the answer to the “pie in the sky” Example 2, maybe you can be convinced that in the general case, given $p_S$, the mean value $\mu_G$ of a geometrically distributed random variable $X_G$ is:

$$\mu_G = \frac{1}{p_S}. \quad (5.8)$$

**Observation 8 (Uninteresting $p_S$ values)** Note that if $p_S = 1/2$, then the mean value $\mu - q$ is 2. If $p_S$ is greater than 1/2, then $\mu$ is between 1 and 2.

As $p_S$ approaches value 1, most of the experiments end on the first trial.

**Remark 2 How boring!**

For this reason, the interesting case for r.v. $X_G$ is when the probability that the failure probability $p_S$ should be the larger probability.

5.4.2 Geometric Series and the Geometric Distribution

A geometric series takes the following form, where $a$ is constant, and $q$ is any real number:

$$a + aq + aq^2 + aq^3 + aq^4 + \ldots, \neq 0. \quad (5.9)$$

In general, the form is:

$$\sum_{i=0}^{\infty} aq^i$$

The interesting case is when $-1 < q < 1$, in which case the sum converges to:

$$\sum_{i=0}^{\infty} aq^i = \frac{a}{1-q}. \quad (5.10)$$

For Eq 5.10, let the numerator $a$ be 1, and $q$ be the probability $p_S$ that stops the Bernoulli experiment. See the discussion surrounding Eq 5.8.
The application of the geometric distribution to run length coding is most advantageous for long runs.\footnote{If runs are generally less than 2 or 3, other coding techniques may apply.} The larger the value of the probability the run (continues), denoted $p_c$, the better the performance of run length coding. And so the probability of stopping the experiment, $p_s$, should be low. Therefore, we wish $p_c = 1 - (p_s) > 0.5$.

Let $x$ be the value of the geometrically distributed random variable $X_G$ for an experiment. Then:

$$p(x) = p_s \times p_c^{x-1}, \quad (5.11)$$

for $x = 1, 2, \ldots$, and undefined if $x$ is any other value.

In Eq 5.11, mathematicians call $p_s$ the probability of success, whereas in run length coding we call it the probability the run stops. Also, the Geometric distribution is based on the assumption of independence from one Bernoulli trial to the next. The probability $p(x)$ that the experiment ends on the $x^{th}$ trial in Eq 5.11 is defined as the product of the respective probabilities of $x-1$ failures, and 1 success. (The first success stops the experiment).

The random variable $x$ of the geometric distribution is the number of trials to the first success, and is characterized by Eq 5.12. Clearly Equation 5.12 shows 1 success trial and $x - 1$ failed trials ($x = 1 + (x-1)$), since the probability of $x$ trials is the product of the probabilities of each trial outcome. Replacing $x$ by 1, 2, ..., etc., or the $x$-axis, and evaluating $p(x)$ for the y-axis, we have a histogram or bar chart of the geometric distribution for any particular value of $p_s$ we choose, since for each Bernoulli trial $p_s + p_c = 1$.

$$p(x) = p_s \times p_c^{x-1} \quad (5.12)$$

Suppose we are interested in summing the probabilities for $x = 1, 2, \ldots, m-1$. We can also solve the problem by determining the probability that random variable $x$ equals or exceeds value $m$, and subtracting that result from 1. Taking Eq 5.10 and Eq 5.12 together, assuming $p_s$ is known, we can determine the probability that the geometric distribution’s random variable $X_G$ equals or exceeds some integer value $md$. Probability $p(x = m) = p_s \times p_c^{m-1}$ is the first term. Factor $p_s$ is common to all terms, and the exponents for $p_c$ is increased by 1 for each successive term, beginning with exponent $md - 1$.

$$p(X_G \geq m) = p(X_G = m) + p(X_G = m + 1) + p(X_G = m + 2) + p(X_G = m + 3) + \ldots$$

$$= p_s \sum_{i=m-1}^{\infty} p_c^i \quad (5.13)$$

Substituting Eq 5.10 for the infinite sum (from 0 to $\infty$), where $q$ is identified by $p_c$, and constant $a$ is 1, we have:

$$p(x \geq m) = p_s \times p_c^{m-1} \frac{1}{1 - p_c} \quad (5.14)$$

Note that $p_s = (1 - p_c)$, and factor $p_s$ in the numerator cancels the value of the denominator:

$$p(x \geq m) = p_c^{m-1} \quad (5.14)$$

One assumption for the geometric distribution is that $p_c$ does not change as the run length increases. In other words, no matter how long the run becomes, the probability $p_c$ that the run continues remains the same.

### 5.5 Run length coding and the geometric distribution

The run length random variable $R$ is defined as the number of times the run symbol occurs before some other symbol ends the run. In run length coding, the symbol that ends the run corresponds to the event that ends the experiment of the geometric distribution.

In practical terms, the differences between the geometric distribution random variable $X_G$, and run length random variable $R$ are listed below.

1. The geometric random variable $X_G$ does count the final Bernoulli trial that ends the Bernoulli experiment.
2. The run length random variable $R$ counts only the number of times the run continues, and does not count Bernoulli trial that terminates the run.
3. If the experiment ends after the first Bernoulli trial, the value $x$ of random variable $X_G$ is 1, and value $r$ of random variable $R$ is 0.
4. The relation between the outcome $r$ of run length variable $R$ and outcome $x$ of the geometric distribution variable $X_G$, for the same experiment is: $r = x - 1$.
5. In run length coding:

$$p(r) = p_s \times p_c^r \quad (5.15)$$

6. In the geometric distribution, $p(x) = p_s \times p_c^{x-1}$
For $R$, if success is achieved on the first trial, the outcome is 0. A run length $R$ of 0 means we were looking for the first instance of the designated run symbol, but the run stop event $S$ occurred.

If the symbol alphabet is binary, then the value of the event that stopped the run is known to be the second symbol. In any case, binary or M-ary alphabet, we can use $p_S$ for the run symbol, and $p_R$ for any other, such that $p_R = 1 - p_S$.

The scenario for a run mode within a data string from an M-ary alphabet $A$ is simply a sequence of binary events; one of two atomic events: $C$ and $S$. The run symbol is $C$ and any other symbol $S$. If we are doing runs within an M-ary alphabet, then the event of $S$ should be followed by the non-run member of the alphabet that stopped the run. Curiously, in using run length coding in M-ary alphabets, and triggering “run mode” with two successive symbols of the same value, one can achieve a compression gain even if the probability of the designated run symbol is less than $0.5$. The reason is that one saves enough bits when the run symbol repeats to pay of the cost of being wrong the majority of the time. If “run stop” is greater than $0.5$ then with arithmetic coding, it costs less than a bit to be wrong, and then code the unknown symbol under the probability one would have used anyway. On the other hand, if the run symbol occurs, then no more bits are needed.

5.6 The Family of Golomb Codes

In the previous section, we indicated the difference between the geometric distribution random variable $x$ and run length variable $r$ as $x = r + 1$, since $r$ only counts the run continues event of probability $p_S$.

In Golomb’s [Gol66] algorithm, each codeword corresponds to a run length. The binary coder receives a binary input sequence of symbols, or simply input. We can treat a sequence of 0s and 1s as binary events $C$ and $S$, where $C$ continues the run and $S$ stops the run. The probability of the run length random variable value 0, 1, 2, ..., etc., is a distribution characterized by the single parameter $p_S$: the “run continues” probability. The “run stop” probability, $p_R$, is calculated as $1 - p_S$.

A sequence of these binary events may be parsed into run lengths of zero or more $C$ (or Continue) events followed by a single $S$ (or Stop) event.

The key to Golomb’s parameterized family of codes, is that Golomb converts ranges of probability value $p_C \geq 1/2$ to a single parameter characterized by a positive integer $md \geq 1$, such that the following approximation holds, where $md$ is an integer that we call the Golomb number (or parameter):

\[ p_C^m \approx \frac{1}{2} \quad (5.16) \]

We show that the value of the Golomb number $md$ approximates the median value (subsequently denoted as $md$ in what follows) for the run length distribution.

The median of the distribution of a random variable is the numeric value such that approximately half of the random variable occurrences have a larger value, and the remaining occurrences have a lesser value.

For many distributions, particularly those that are symmetric about the mean, the mean value and median value are the same. However, for skewed distributions such as the geometric distribution, the mean and median values are different. The median value is that value such that if one draws a relatively large sample runs from the the distribution, and then one sorts the sample values according to their magnitude, one discovers that half the samples are above the median value and half below.

Put another way, if one draws a single sample at random, there is about a 50% chance the sample value is equal to or less than the median, and about a 50% chance the sample value is equal to or greater than the median. In other words, the probability mass above the median value is the same as the probability mass below the median value. One can devise examples, however, for which the median value does a poor job of equally dividing the distribution.

For the median run length, we examine Eq 5.15; for each run $r$ probability $p(r)$ to find run length value $r(md)$ such that:

\[ p(r < md) = \sum_{i=0}^{md} p_S \times p_C^i \approx \frac{1}{2}. \quad (5.17) \]

Alternatively, since the probability mass less than the median $md$ should approximate the probability mass greater than the median:

\[ p(r > md - 1) = \sum_{i=md}^{\infty} p_S \times p_C^i \approx \frac{1}{2}. \quad (5.18) \]

Starting with $md$, the first few terms are:

\[ p(r > md - 1) = p_S \times p_C^{md} \\
+ p_S \times p_C^{md+1} \\
+ p_S \times p_C^{md+2} + \cdots \quad (5.19) \]

First, factor $[p_S \times p_C^{md}]$ from Eq 5.19:

\[ p(r > md - 1) = \]

\[ p_S \times p_C^{md} \times [1 + p_C + p_C^2 + \cdots] \quad (5.20) \]

Refer to constant $a$ in Eq 5.10, and note that in Eq 5.20 the arbitrary constant $a$ is 1. Also recognize the right factor enclosed in brackets ([1]) of Eq 5.20 is
the sum of a geometric series. Replacing the bracketed factor with its closed form \( \frac{1}{1 - p_c} \):

\[
p(r > md - 1) = (ps \times p_c^{md}) \times \frac{1}{1 - p_c}
\]

\[
= (ps p_c^{md}) \times \frac{1}{p_c^s}
\]

\[
= p_c^{md} \approx \frac{1}{2}. \quad (5.21)
\]

Compare Eq. 5.16 and Eq. 5.21, and marvel that the median run length is indeed the Golomb number \( md \).

Note also that the median value for the geometric distribution’s random variable is \( m+1 \), because for a given \( p_c \), the number of Bernoulli trials per experiment is one greater than the run length value for the same experiment.

Gallager and VanVoorhis [GVV75] show the Golomb code to be optimal (assuming the constraint of an integer-length codeword for each run), and characterize its coding efficiency. Their work provides the equation that defines the breakpoint probabilities that bound the probability ranges applicable to each value of Golomb parameter \( md \), making precise the probability range for each value of \( md \).

**The Golomb code: predetermined \( ps \)**

The Golomb Code is actually a family of codes. Each member of the family has a parameter \( md \) (whereas, Golomb used notation \( m \) in his original paper). In a sense, Golomb quantizes the values of \( ps \) into probability ranges and assigns a value of \( md \) to each such range. Here, we call the value of parameter \( md \) the Golomb number that distinguishes each family member.

Golomb designed a family of countably infinite Golomb codes, each such code based on positive Golomb number \( md \), and being optimal for its own nonoverlapping probability range. The first range, or **unity range**, corresponds to \( md \)-value 1 and is valid for values of \( ps \) or \( p_c \) in the range \([0.5, 0.38196601]\), or which is the same, range \([0.61803399, 0.38196601]\). The second range \((m = 2)\) is valid for values \( ps \) in the range \([0.38196601, 0.245123336]\). As implied by the notation, the ranges are closed on the high-valued probability boundary, and open on the low-valued boundary and so are nonoverlapping.

The breakpoint probabilities, i.e., the values separating the quantized range of the binary probabilities, are transcendental numbers. The range for \( md = 1 \) is approximately from 0.618 to 0.832.

Corresponding to each Golomb number \( md \) and run length \( r \) is a two-component codeword.

Defining a Golomb code, given \( md \) and run length \( r \):

1. One component is a unary or base \( \lfloor \frac{1}{md} \rfloor \) (also called stone-age binary) \(^3\) encoding of the integer-valued quotient \( Q \) of \( r \mod md \).
2. The second component is the prefix codeword encoding of the integer remainder \( R \): \( r \mod md \).

The run length \( r \), for a given \( md \), thus deals with a quotient \( Q \), \( \lfloor \frac{r}{md} \rfloor \), and remainder \( R = r - md \times Q \). The code naturally has two parts \( Q \) and \( R \), and in fact uses the same unary code for \( Q \) independent of the binary probability distribution for events \( C \) and \( S \). The remainder \( R \) is encoded by a code that is dependent upon the Golomb parameter \( md \). When \( md \) is a power of \( 2 \) (e.g., \( 2^k \)), then \( R \) must be 0, 1, \ldots, \( k-1 \), and so there are \( md = 2^k \) codewords, each \( k \)-bits in length. For example, when \( md \) is \( 4 = 2^2 \), the four codewords for \( R \) are 00, 01, 10, and 11.

The decoder can reconstruct the run length value \( r \) by multiplying Golomb number \( md \) by the quotient \( Q \), and then adding the remainder \( R \):

\[
r = (md \times Q) + R. \quad (5.22)
\]

The Golomb code encodes quotient \( Q \) as a unary (number) code) of the form zero or more 1s followed by a delimiting 0 that signals the end of the unary number for \( Q \). The \( md \) possible values of the remainder \((0, 1, \ldots, md-1)\) are almost equally likely with the lower value slightly more popular than the higher, as in the geometric distribution.

**Golomb Codes when \( md \) is \( 2^k \)**

The most difficult part of creating the Golomb code concerns the codeword for the remainder \( R \). For this part, we need to encode values of 0, 1, 2, \ldots, \( md-1 \).

We first consider value \( md \) of the form \( 2^k \) where \( k \) is an integer; 1, 2, 3, 4, etc. These special values of \( md \) are the “power of 2” values; 1, 2, 4, 8, etc. In this special case, the code for \( R \) is the \( k \)-bit binary number representing the remainder \( R \).

For example, if \( k \) is 1, then \( md \) is 2, and the remainder values are 0 and 1. These are represented by the 1-bit values (since \( k=1 \)) of 0 and 1. If \( md \) is 4, then the values of \( R \) are 0, 1, 2, and 3, and are respectively coded as 00, 01, 10 and 11. In contrast, if \( md \) is 3, the codes for the remainder values of 0, 1, and 2 are respectively 00, 10, and 11.

To show the basic idea of the Golomb code, in this section we treat the case of the simple code for \( R \) where residual is a \( k \)-bit binary number. When \( md \) is a power of two, i.e., has value \( 2^k \) where \( k = 0, 1, 2, \ldots \) etc., the code for remainder \( R \) is straightforward.

Example: \( md = 2, \) run \( r = 5 \).

**To Encode:**

\(^3\)Early number systems used a vertical stroke to record each instance.
1. \( Q = \lfloor 5/2 \rfloor = 2 \). Converting \( Q = 2 \) to the unary code: 110. With two 1s preceding the delimiter 0, we encode the number 2. This means \( md = 2 \) divides \( r = 5 \) twice.

2. With \( md = 2 \), the remainder set is \( \{0, 1\} \). In this case:
   - \( \mathcal{R} = r - mQ = 5 - (2)(2) = 5.4 = 1 \).

Result: 1101

Had the run value \( r \) been 4, then remainder \( \mathcal{R} \) would be 0, and the code would be 1100.

To Decode:

1. The first 0 of 1101 is preceded by two stone-age 1s, so \( Q = 2 \).
2. Following 110, we have 1, so remainder \( \mathcal{R} \) is 1.
3. From Equation 5.22, \( r = (2 \times 2) + 1 = 5 \).

Let \( k \) be an integer so when \( md \) is of the form \( 2^k \), then the mod \( md \) remainder is encoded as a \( k \)-bit integer. When \( md \) is such that \( 2^k < md < 2^{k+1} \), then at least one codeword has \( k \) bits and the others \( k + 1 \) bits.

In [GVV75, BK74] the run length encoding algorithm is implemented with the aid of a module-\( md \) counter that behaves as follows. The modulo-\( md \) counter is reset to 0 at the start of each new run. The counter counts instances of the run continues outcome. If the count goes from \( md - 1 \) to 0, the counter has rolled over.

Golomb Code Algorithm, “Recursive” version

Here the recursion is based on Bernoulli trials whose two outcomes are \( C \) (run continues) and \( S \) (run stops).

Initialization: Fix Golomb parameter value \( md \) to \( 2^k \).
Clear (or reset) Mod-\( md \) Counter to 0, including the high-order Carry-bit value in bit position \( 2^k \). Clear binary-valued Run-Stop Flag to 0.

The Run-Stop Flag is set to 1 when the run-stop symbol (S) occurs. The Run-Stop Flag, when set, remembers that the quotient part or unary code for the Run length has been done and concatenates the delimiter 0.

Having encountered the Stop symbol, and treated the unary code part of the Golomb code, means that value \( Q = \) number of times Golomb parameter \( md \) divides the Run length value has been encoded. The final step produces the prefix code for the Run length’s remainder value \( \mathcal{R} = \) Run length mod \( md \). The remainder value \( \mathcal{R} \) is the value of the \( k \) least significant bits of the Mod-\( md \) Counter at the time the Stop symbol is processed. The final step of the encoder left-shifts the least-significant \( k \) bits of that counter to the code string \( Cd(S) \).

Mod-\( md \) Counter operation, when \( md \) is a power of 2, say \( 2^k \), the Mod-\( md \) counter counts from 0 to \( 2^k - 1 \). When this value in the Mod-\( md \) counter is \( 2^k - 1 \), and is then incremented, the Mod-\( md \) counter rolls over to 0, and sets a Carry signal in the bit position of weight \( 2^k \).

In the Encoder Algorithm below, it may be the case that the sequence ends before a Stop symbol occurs. Thus there are three conditions: handle the \( C \) symbol, handle the \( S \) symbol, and handle an End-of-File condition.

Encoder Algorithm

1. Read next symbol.
2. If run continue symbol \( C \),
   (a) Set Run-End Flag to 0.
   (b) Increment (Mod-\( md \)) Counter. If Counter Carry is set:
      - Concatenate value 1 onto the codestring.
   (c) Return to Step 1.
3. Else if run stops symbol \( S \) occurs, then:
   (a) Concatenate delimiting value 0 to the codestring, ending the unary code for \( Q \).
   (b) Concatenate the value of the \( k \)-bit (module-\( md \)) Counter to the codestring.
   (c) Mod-\( md \) Counter is reset to 0.
   (d) Run-End Flag = 1.
   (e) Return to Step 1.
4. Else if end-of-file occurs,
   (a) If Run-End Flag is 1, go to Exit (step 4d). Otherwise continue:
      (b) Concatenate value 0 to the codestring.
      (c) Concatenate the \( k \)-bit Mod-\( md \) Counter value to the codestring.
      (d) Exit the coding algorithm.

Encoder Example: \( md = 2 \), source string \( S = 111110 \), where binary value 1 represents event \( C \) (continue) and binary value 0 represents event \( S \) (stop).

Initial Conditions: Counter = 0, Run-end Flag = 1.
For the Read operation, we read input 111110 one symbol at a time. Symbol 1 is the Continue symbol \( C \) and 0 the Stop symbol \( S \). The Mod-\( md \) Counter is simply denoted “Counter” in what follows.
1. Read (first) 1. Increment Counter. Counter = 1. 
   Run-end Flag = 0.
2. Read (second) 1. Increment Counter. 
   Counter Carry bit is set: Concatenate 1 to Cd A, so now 
   Cdi = 1. Counter reset to 0. Run-end Flag = 0.
3. Read (third) 1. Increment Counter. Counter = 1. 
   Run-end Flag = 0.
4. Read (fourth) 1. Increment Counter. 
   Counter Carry bit is set: Concatenate 1 to Cd 1 so Cdi = 11. 
   Counter reset to 0. Run-end Flag = 0.
5. Read (fifth) 1. Increment Counter. Mod-m 
   Counter = 1. Run-end Flag = 0.
6. Read 0. Concatenate delimiting 0 to Cdi 11, so 
   Cdi = 110. Concatenate Mod-m Counter 1 to Cdi 110, so Cdi = 1101. Reset Mod-m Counter. 
   Run-end Flag = 1.
7. Read End-of-File. Run-end Flag is 1, so Exit.

In this case, the data string ended with a Run. Suppose otherwise. What do we do when the file ends 
with the run character? The answer to this question 
depends on the some “practical issues of compression”. 
One such practical issue concerns:

- “How does the decoder know it has decoded the 
  last character?”

- One of the answers: tell the decoder how many 
  data characters to be decoded. Stop reconstruct-
  ing the data string after decoding that many char-
  acters.

The Golomb coder algorithm must terminate the 
codestring if the End of File is received. If the coder 
handles one or more “run continues” symbols at 
that point, then we “make believe” that we received 
a “run stop” symbol. To implement this idea, the de-
limiting 0 is concatenated to the end of the codestring, 
and then the mod-md Counter value is placed on the 
codestring (following the delimiting 0).

For example, suppose string S is 11111011, which 
do not ends with the run stop character. Suppose 
we tell the decoder that the file has 8 symbols, and 
the decoder decodes 9 (a run of length 5 and a run of 
length 2: 11111011). The decoder only reconstructs 
the first 8 symbols, 11111011, which is correct.

It turns out that for this run length algorithm, and 
for many other compression algorithms, the decoder 
needs additional information to indicate when to stop 
decoding. A good strategy is to communicate the 
length of the original data string to the decoder. 
Reserving a length field in the header of the code string is 
a good way to tell the decoder the size of the original 
data string.

Dealing with values of $md$ that are not powers of 2

If $md$ is greater than $2^k$ but less than $2^{k+1}$, then 
$2^k < md < 2^{k+1}$. Since $2^k + 2^k = 2^{k+1}$ depending 
on how many k-bit codewords we extend to two (k+1)-
bit codewords, we can adjust the number of (k+1)-bit 
codewords to fit all the additional remainder values 
for the larger Golomb value $md$ that fits between $2^k$ 
and $2^{k+1}$.

Consider an example. If $m=5$, then $2^k < 5$ is 4, and 
and $k$ is 2. Consider the 2-bit codewords for the remainder 
values from $md = 2^k = 4$: 0 is 00, 1 is 01, 2 is 10, and 3 
is 11. Let h denote the number of codewords we need 
both the $2^k$: $h = md - 2^k$. In this example 5-4 = 
1 more codeword. So, to get a 5th needed remainder 
value codeword beyond the 4 for $m=4$, for $md = 5$, 
we have only to extend the codeword in the last bit 
position: 11 extends to 110 and 111.

So, we represent all 5 remainder values for $md = 5$ 
as follows: 0 as 00, 1 as 01, and 2 as 10 (these 3 as in 
the m=4 case). And the new codewords for $m=5$ are: 
3 as 110, and 4 as 111. Note that 3 was represented by 
11 before: if we split the codeword for remainder value 
3, we still need to represent 3 with a longer codeword.
This observation is true in general.

Now consider $md = 6$. Here $h = 6-4 = 2$, so 2 of 
the k=2 length codewords need to be split, so that 
$2h = 4$ codewords of length 4 are needed. The two 
length 2 codewords are the last two: 10 and 11, which 
respectively become 100 and 101 from 10, and 110 and 
111 from 11.

Therefore 2-bit codewords 00 and 01 continue to re-
present remainder values 0 and 1. This leaves (in mag-
nitude order) codeword 100 for a remainder value of 
2, 101 for 3, 110 for 4, and 111 for 5.

In summary, of the longer codewords, half will rep-
resent remainder values that were shorter codewords 
in the 2^k Golomb code case: the other half will repre-
sent the h codewords for the $h = md - 2^k$ remainder 
values that exceed remainder whose value $2^k - 1$.

Now consider the more general case. Let Golomb 
value $md$ exceed $2^k$ by value h: $md = h + 2^k$. We need 
h more codewords. Then there will be 2h codewords of 
length k+1, because we need to represent the h 
reminders these h codewords formally represented, 
plus the h new remainder values above the $2^k$ for which 
$2^k < Golomb md$. The codewords are assigned in 
magnitude order: the lower the value of the remainder, 
the smaller that value of the assigned codeword.

As another example, if $md = 13$, then 8 < 13 < 16. 
If $md$ were 8, all codewords would be 3 bits and if $md$ 
were 16, all codewords for the remainder would be 4 
bits. In this case, $md$ is 5 larger than 8 (13 - 8 = 5), 
so 5 of the 3-bit codewords become 4-bit codewords, 
since only 3 3-bit codewords remain to represent the first 
3 remainder values 0, 1, and 2. The other 13 - 3 = 10 
remainder values are represented 4-bit codewords. So 
we have 3 3-bit codewords and 10 4-bit codewords for

CE 263 Winter 1999
the 13 remainder values 0, 1, ..., 12, which account for the \( md = 13 \) codewords needed.

In general, if \( 2^k < md < 2^{k+1} \), there are \( (md - 2^k) \) codewords of length \( k + 1 \) that are paired and serve \( 2(md - 2^k) \) or \( md - 2^{k+1} \) code words of \( k + 1 \) bits each. We need \( md \) codewords overall, so \( md - 2md + 2^{k+1} = 2^{k+1} - md \) code words must have \( k \) bits. The number of code words are: \((2md - 2^{k+1}) + (2^{k+1} - md) = md.4\)

To construct the 13 remainder (mod 13) codes, for values 0, 1, and 2 are the 3-bit binary numbers for these values: 000, 001, and 010. Values 3 through 12 use the remaining code space, beginning with 011 which is next, with one bit position added to the right of each of the remaining 5 3-bit codewords, so they become 4-bit codewords. The result is that remainder value 3 is assigned 0110, 4 gets 0111, 5 gets 0000, 6 gets 0010, 7 gets 0100, 8 gets 1000, 9 gets 1001, 10 gets 1010, 11 gets 1100, and 12 gets 1111. Note in all cases that the largest remainder value, which is m-1, consists of all 1s and is the largest code word for the remainders.

**Examples**

**Example.** \( md = 5 \).

Notice that the space between the “unary” part and the “remainder” part is there for better understanding and readability of the code.

<table>
<thead>
<tr>
<th>Run</th>
<th>Codeword</th>
<th>Run</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 00</td>
<td>13</td>
<td>110 110</td>
</tr>
<tr>
<td>1</td>
<td>0 01</td>
<td>14</td>
<td>110 111</td>
</tr>
<tr>
<td>2</td>
<td>0 10</td>
<td>15</td>
<td>111 00</td>
</tr>
<tr>
<td>3</td>
<td>0 110</td>
<td>16</td>
<td>111 01</td>
</tr>
<tr>
<td>4</td>
<td>0 111</td>
<td>17</td>
<td>111 10</td>
</tr>
<tr>
<td>5</td>
<td>1 00</td>
<td>18</td>
<td>111 110</td>
</tr>
<tr>
<td>6</td>
<td>1 01</td>
<td>19</td>
<td>111 111</td>
</tr>
<tr>
<td>7</td>
<td>1 10</td>
<td>20</td>
<td>11110 0</td>
</tr>
<tr>
<td>8</td>
<td>1 110</td>
<td>21</td>
<td>11110 01</td>
</tr>
<tr>
<td>9</td>
<td>1 111</td>
<td>22</td>
<td>11110 10</td>
</tr>
<tr>
<td>10</td>
<td>110 0</td>
<td>23</td>
<td>11110 110</td>
</tr>
<tr>
<td>11</td>
<td>110 1</td>
<td>24</td>
<td>11110 111</td>
</tr>
<tr>
<td>12</td>
<td>110 10</td>
<td>25</td>
<td>111110 0</td>
</tr>
</tbody>
</table>

Notice above where values of \( R \) of 0, 1, and 2 have a 2-bit code, and values of \( R \) of 3 and 4 have a 3-bit code.

**Example.** \( md = 13 \).

<table>
<thead>
<tr>
<th>Run</th>
<th>Codeword</th>
<th>Run</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 000</td>
<td>14</td>
<td>10 001</td>
</tr>
<tr>
<td>1</td>
<td>0 001</td>
<td>15</td>
<td>10 010</td>
</tr>
</tbody>
</table>

\(^4\)Thanks to Lawrence Searcy for this observation.

<table>
<thead>
<tr>
<th>Run</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0 010</td>
</tr>
<tr>
<td>3</td>
<td>0 110</td>
</tr>
<tr>
<td>4</td>
<td>0 111</td>
</tr>
<tr>
<td>5</td>
<td>0 100</td>
</tr>
<tr>
<td>6</td>
<td>0 001</td>
</tr>
<tr>
<td>7</td>
<td>0 101</td>
</tr>
<tr>
<td>8</td>
<td>0 111</td>
</tr>
<tr>
<td>9</td>
<td>0 110</td>
</tr>
<tr>
<td>10</td>
<td>0 101</td>
</tr>
<tr>
<td>11</td>
<td>0 110</td>
</tr>
<tr>
<td>12</td>
<td>0 111</td>
</tr>
<tr>
<td>13</td>
<td>1 000</td>
</tr>
</tbody>
</table>

| Value | I am the run continues (re) symbol, and value 0 is the run stop (rs) symbol. Let \( md = 5 \).
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Codeword</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>In:</td>
<td>1 1 1 1 1 1 1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>Ctr:</td>
<td>0 1 2 3 4 5 1 2 3 4 5 0 1 2 3 4 4</td>
</tr>
<tr>
<td>Out:</td>
<td>1 1 0 1 1 1</td>
</tr>
</tbody>
</table>

\[ \text{Codeword: 110111.} \]

To decode the codeword: 110 (5\(x\)2) + 111 (remainder of 4): run length 10+4 = 14.

**Determination of Golomb \( md \)**

The selection of Golomb number \( md(p_c) \) has an ingenious explanation that undoubtedly motivates its discovery. We know that if the probability of an event is \( \frac{1}{2} \), then it should be encoded with a single bit. Working backwards from that notion, we examine the unary part of the code and ask: to add one bit to the unary code, what should the probability be? The answer: we should encode as many instances of the “run continues” outcome, say \( md \) of them, each at probability \( p_c \), such that \( p_c^m = \frac{1}{2} \).

Put another way, suppose that in run length coding we discover it’s optimal when \( md \) successive C events should increase the code length by one bit. In other words, suppose that \( md \) successive C events have a self-information of 1 bit. Converting the self-information to a probability, this means that \( md \) successive C events should have a probability of \( \frac{1}{2} \). Golomb’s response has the insight of defining parameter \( md(p_c) \) to be such that \( p_c^m = \frac{1}{2} \). Taking log to base 2 of each side, we have \( md \log_2 p_c \approx -1 \). Solving for \( md \) value:

\[ md \approx -1 \frac{1}{\log_2 p_c}. \quad (5.23) \]
Gallager and Van Voorhis [GVV75] have proven the Golomb code is optimal for integer-length code words, where Golomb number \( md \) is the unique positive integer that satisfies inequality Eq 5.24 for the design probability \( p(C) \). Eq 5.24 sharpens Golomb’s Eq 5.23 and identifies the precise breakpoint probability values \( p_c \) that separate successive Golomb numbers \( md \) and \( md + 1 \).

\[
p_c^{md} + p_c^{md+1} < 1 < p_c^{md} + p_c^{md-1} \quad (5.24)
\]

Table 5.1 shows properties of the Golomb code. Eq 5.24 of Gallager and VanVoorhis is used to identify ranges of \( \text{run stop} \) probabilities. The points were calculated experimentally in the APL programming environment for a given value of \( md \) by using test values of \( p \) until equality was almost reached in the left half of Eq 5.24.

### Table 5.1: Range of \( p_R \) and \( \mu(R) \)

<table>
<thead>
<tr>
<th>Golomb Probability range ( md ) for ( \text{run stop} p(\text{Gr}) )</th>
<th>( E(R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1 ( 0.618034 ) – ( 0.381966 )</td>
<td>0.6 - 1.6</td>
</tr>
<tr>
<td>2 ( 0.381966 ) – ( 0.246122 )</td>
<td>1.6 - 3.1</td>
</tr>
<tr>
<td>3 ( 0.245122 ) – ( 0.180827 )</td>
<td>3.1 - 4.5</td>
</tr>
<tr>
<td>4 ( 0.180827 ) – ( 0.143325 )</td>
<td>4.5 - 6.0</td>
</tr>
<tr>
<td>5 ( 0.143325 ) – ( 0.118729 )</td>
<td>6.0 - 7.4</td>
</tr>
<tr>
<td>6 ( 0.118729 ) – ( 0.101346 )</td>
<td>7.4 - 8.9</td>
</tr>
<tr>
<td>7 ( 0.101346 ) – ( 0.088408 )</td>
<td>8.9 - 10.3</td>
</tr>
<tr>
<td>8 ( 0.088408 ) – ( 0.078401 )</td>
<td>10.3 - 11.7</td>
</tr>
<tr>
<td>9 ( 0.078401 ) – ( 0.070430 )</td>
<td>11.7 - 13.2</td>
</tr>
<tr>
<td>10 ( 0.070430 ) – ( 0.063931 )</td>
<td>13.2 - 14.7</td>
</tr>
</tbody>
</table>

Column E\((R)\) of Table 5.1 is calculated as follows. For each value of \( p_R \), we performed the following calculation: \( \frac{1}{p} \) to obtain the mean \( \mu \) of the geometric distribution, then subtracted 1 to obtain the mean run length: \( \mu_R \).

Example: Suppose we wish to encode run lengths 7, 13, 8, 6, and 11. In this case, we have 5 “run-stop” characters, and \( 7 + 13 + 8 + 6 + 11 = 45 \) “run continues” characters. That totals 50 characters. So \( p_c = \frac{45}{50} = 0.9 \). Therefore \( p_R = 0.1 \). By trial and error we can use Eq 5.24 with various values of \( md \) to determine the value that satisfies the equation for \( p_R = 0.9 \). We satisfy Eq 5.24 if we let \( md = 7 \).

\[
p_c^{7} + p_c^{8} = 0.908764, \text{ and } p_c^{7} + p_c^{8} = 1.00974.
\]

As a check, note that \( p_R = 0.1 \) agrees with Table 2 (of Breakpoints) for \( m = 7 \), since 0.088 ≤ 0.1 < 0.101.

With the proper value \( md \) of 7, we need to determine the codes for the seven remainder values of 0, 1, 2, 3, 4, 5, and 6. It turns out that, were there 8 remainder values, each would be of 3 bits (000, 001, 010, ..., 110, 111). With 7 values, it means the first value can be of only two bits: 00 (the union of 000 and 001). So the next six values are 010, through 111.

The Code Table for Remainder values \( R \) is:

<table>
<thead>
<tr>
<th>Run</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>011</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>110</td>
</tr>
<tr>
<td>6</td>
<td>111</td>
</tr>
</tbody>
</table>

The encoding of runs 7, 13, 8, 6, 11 proceeds as follows: 7: 1000, 13: 10111, 8: 10010, 6: 0111, 11: 1101.

Concatenating the individual runs onto a single codestring, we have:

1000101111001001110101

which is 23 coded bits instead of 30 uncoded bits.

Now let us decode the runs. The decoding procedure is:

1. Parse off the leading 1s (which may be none) up to and including the first delimiting 0 that ends the unary code used for \( Q \).
2. Set \( Q \) equal to the number of leading 1s.
3. The value \( R \) can be determined from a Decode Table.

   (a) If \( md \) is a power of 2, eg., \( 2^k \), then the \( k \)-bit binary number is the value of \( R \). Otherwise,

   (b) \( 2^{k-1} < md < 2^k \), and a \( k \)-bit Decode Table can be used to decode \( R \).

4. Once \( Q \) and \( R \) are determined: \( r = (md \times Q) + R \).

For the Decode Table of the example, we have:

<table>
<thead>
<tr>
<th>Address</th>
<th>Run</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>010</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>011</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>101</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>110</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>111</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

To aid the understanding, we repeat the codestring below, but place a comma after each run, and underscore the the “0” that delimits the unary part for \( Q \) from the remainder part \( R \).
Note that when 2 is followed either by 000 or by 001, the remainder part R represents only two bit positions (00) in the code.

We proceed to decode 1000101111100101110101 step by step:

1. First run, step 1, pulls off 10 so $Q_1$ is 1, and we have 0011011100101110101. The first 3 bits of what remains is the address 001 that gives $R_1$ of 0, and shift out 2 bits. $r_1 = (7 \times 1) + 0 = 7$.

2. Remaining code: 101111001001110101. Second run, step 1, pulls of 10 again, so $Q_2$ is 1, and we have 11110010001110101. The decode table decodes $R_2$ of 6, and shift out 3 bits. $r_2 = (7 \times 1) + 6 = 13$.

3. Remaining code: 10010011110101. Third run, step 1, pulls off 10, so $Q_3$ is 1, and we have 01001111101. The first 3 bits into the Decode Table are 010, which decode to $R_3$ of 1, and shift of 3 bits. $r_3 = (7 \times 1) + 1 = 8$.

4. Remaining code: 011110101. For Fourth Run, step 1, we pull off the first 0, so $Q_4$ is 0, and we have 11110101 left. Feeding the 3-bit field 111 to the decode table, we have $R_4$ of 6, and shift out 3 bits. $r_4 = (7 \times 0) + 6 = 6$.

5. Remaining code: 10101. For Fifth Run, step 1, we pull off 10, so $Q_5$ is 1, leaving 101. Feeding 101 to the Decode table yields $R_5$ of 4, and shift out 3. $r_5 = (7 \times 1) + 4 = 11$. The code string has been reduced to the null string A. We are done.

The result of the decoding operation is 7, 13, 8, 6, and 11. The original binary string would be the 50-bit string:
111110111111111111010111101111111111111111111110.

Goladap: an adaptive Golomb code

In [Lan83], an algorithm called Goladap is described. Goladap exploits the insight $p(C)^m = 1/2$ as a balance point for adapting the Golomb number either up or down. For geometrically distributed run lengths, the probability that the run length is $md$ or greater is $p(C)^m$. For Golomb codes for binary sequences for which Golomb number $md$ is the correct value, run length $md$ is such that half the runs are expected to be of median length $md$ or more.

It takes $md$ Cs before the code adds a stone-age 1, so the first bit of the Golomb code indicates whether the length is 0 to $md - 1$, or is $md$ or greater. Goladap is particularly simple: if at a particular Golomb number $md$, the length of the run is shorter than expected (the mod m counter does not roll over) then the Golomb number is reduced. If the run is longer than expected (the mod m counter rolls over) then the Golomb number is increased. Thus run length $md$ serves as a test in the algorithm, if the algorithm determines the run exceeds the current value of $md$ then a branch toward a more highly skewed estimate is made. If the run ends then the mod $md$ counter has a count less than $md$ and the algorithm determines a less skewed value for $md$. As did [BK74], and for the same reason of a simple implementation, Goladap employs values of $md$ that are powers of 2. Goladap adapts only during code length increases, thus exploiting a normal operation (the mod m counter) of the coding operation, to do adaptation. Also, adaptation only takes place when a stone-age 1 is added, or the run is ended by the occurrence of the S. The increase of value $md$ adaptively was done by Teuhola [Teu78]; the increase of $md$ combined with the decrease of $md$ by one upon the S occurrence was first done in Goladap [Lan83].
Bibliography


