ENSC 861 – Source Coding in Digital Communications

Golomb-Rice Code

Jie Liang
Engineering Science
Simon Fraser University
JieL@sfu.ca

Outline

- Unary Code
- Golomb Code
- Golomb-Rice Code
- Exponential Golomb Code
- Adaptive Golomb Code
  - Applications in JPEG-LS (Lossless JPEG)
  - Adaptive Run-Length/Golomb-Rice (RLGR) Code

Unary Code (Comma Code)

- Encode a nonnegative integer n by n 1's and a 0 (or n 0's and an 1).

n	Codeword
0	0
1	10
2	110
3	1110
4	11110
5	111110
…	…

- Do we need to store codeword table?
- Is this code prefix-free?
- When is this code optimal?

Implementation

- Encoding:

```
UnaryEncode(n) {
  while (n > 0) {
    WriteBit(1);
    n--;
  }
  WriteBit(0);
}
```

- Decoding:

```
UnaryDecode() {
  n = 0;
  while (ReadBit(1) == 1) {
    n++;
  }
  return n;
}
```

Unary Code (Comma Code)

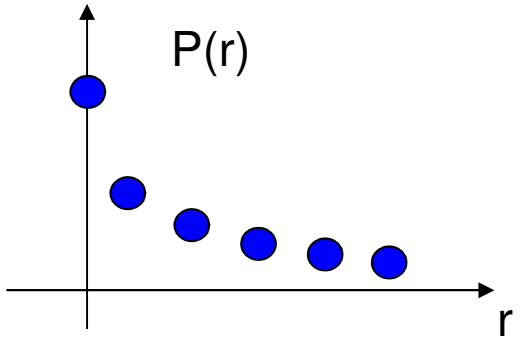
- What if the input has negative integer?

Geometric Distribution (GD)

- Geometric distribution with parameter ρ:
  - $P(x) = \rho^x (1 - \rho)$, $x \geq 0$, integer.
  - Prob of the number of failures before the first success in a series of independent Yes/No experiments (Bernoulli trials).
- ρ = 1/4: P(x): 0.75, 0.19, 0.05, 0.01, 0.003, …
- ρ = 3/4: P(x): 0.25, 0.19, 0.14, 0.11, 0.08, …
- ρ = 1/2:

Why Geometric Distribution?

- Geometric distribution is very useful for image/video compression
- Example 1: run-length coding
  - Binary sequence with i.i.d. distribution
  - $\rho = P(0) \approx 1$:
  - Example: 00000100000000100001100000001
  - Entropy << 1, prefix code has poor performance.
  - Run-length coding is efficient to compress the data:
    - r: Number of consecutive 0's between two 1's
    - ➔ run-length representation of the sequence:
  - Probability distribution of the run-length r:
    - $P(r = n) = \rho^n (1- \rho)$: n 0's followed by an 1.
    - ➔ One-sided geometric distribution with parameter ρ.

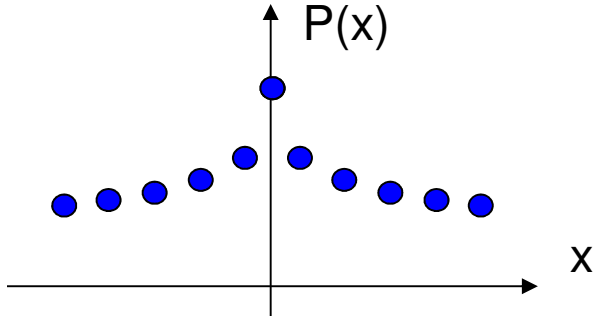


Why Geometric Distribution ?

- Example 2: Prediction error

  e(n) = x(n) – pred(x(1), …, x(n-1)).
- Most e(n)'s have smaller values around 0:
- ➔ can be modeled by two-sided geometric distribution.

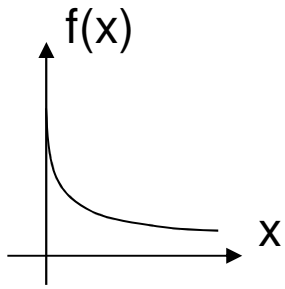
$$p(x) = \frac{1-\rho}{1+\rho}\rho^{|x|}, \qquad 0 < \rho < 1.$$



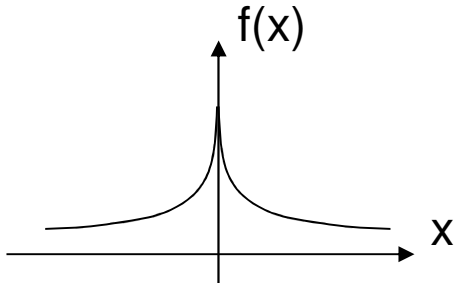
- Two-Sided GD can be approximated by one-sided GD:
  - To reduce the # of contexts of the adaptive entropy coding.
  - Map negative numbers to positive:
    - More than one possible mapping.
    - More on this later (JPEG-LS)

Why Geometric Distribution ?

- GD is the discrete analogy of the Exponential distribution



- Two-sided geometric distribution is the discrete analogy of the Laplacian distribution (also called double exponential distribution)

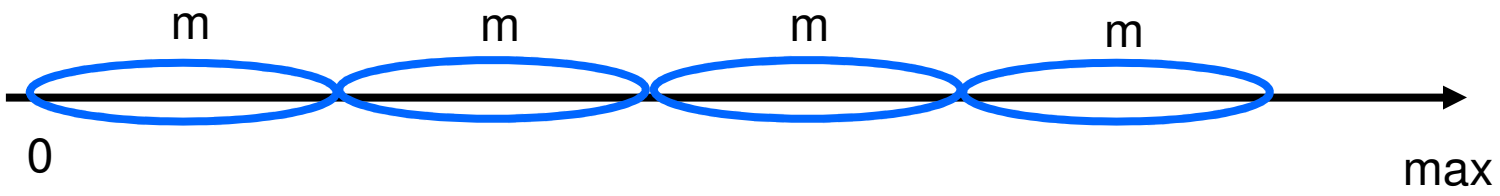


Outline

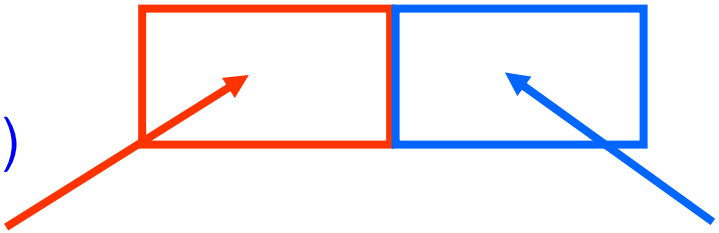
- Unary Code
- Golomb Code
- Golomb-Rice Code
- Exponential Golomb Code
- Adaptive Golomb Code
  - Applications in JPEG-LS (Lossless JPEG)
  - Application in H.264

Golomb Code [Golomb, 1966]

- A multi-resolutional approach:
  - Divide all numbers into groups of equal size m
    - Denote as Golomb(m) or Golomb-m
  - Groups with smaller symbol values have shorter codes
  - Symbols in the same group has codewords of similar lengths
    - The codeword length grows much slower than in unary code



- Codeword :
  - (Unary, fixed-length)



Group ID:
Unary code

Index within each group:
"fixed-length" code

Golomb Code

- q: Quotient, used unary code

q	Codeword
0	0
1	10
2	110
3	1110
4	11110
5	111110
6	1111110
…	…

- r: remainder, “fixed-length” code
- K bits if m = 2^k
  - m=8: 000, 001, ……, 111
- If m ≠ 2^k:

$\lfloor \log_2 m \rfloor$ bits for smaller r

$\lceil \log_2 m \rceil$ bits for larger r

m = 5: 00, 01, 10, 110, 111

Golomb Code with m = 5 (Golomb-5)

n	q	r	code
0	0	0	000
1	0	1	001
2	0	2	010
3	0	3	0110
4	0	4	0111

n	q	r	code
5	1	0	1000
6	1	1	1001
7	1	2	1010
8	1	3	10110
9	1	4	10111

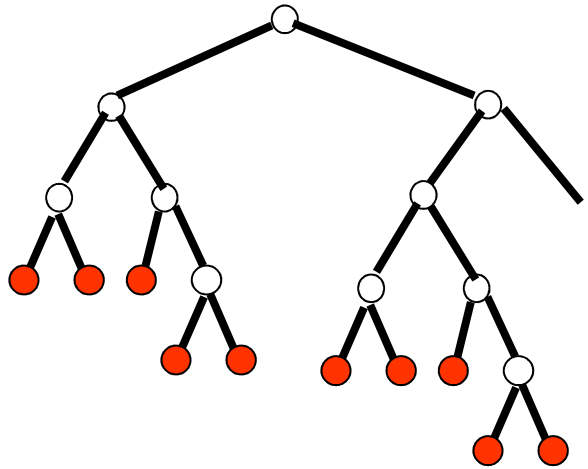
n	q	r	code
10	2	0	11000
11	2	1	11001
12	2	2	11010
13	2	3	110110
14	2	4	110111

Golomb vs Canonical Huffman

- Codewords:

000, 001, 010, 0110, 0111,
1000, 1001, 1010, 10110, 10111

- Canonical form:
  - From left to right
  - From short to long
  - Take first valid spot



- Golomb code is a canonical Huffman
  - With more properties

Golomb-Rice Code

- A special Golomb code with m= 2^k
- The remainder r is the k LSB of n

- m = 8

n	q	r	code
0	0	0	0000
1	0	1	0001
2	0	2	0010
3	0	3	0011
4	0	4	0100
5	0	5	0101
6	0	6	0110
7	0	7	0111

n	q	r	code
8	1	0	10000
9	1	1	10001
10	1	2	10010
11	1	3	10011
12	1	4	10100
13	1	5	10101
14	1	6	10110
15	1	7	10111

Implementation

- Encoding:

Remainder bits:
RBits = 3 for m = 8

```
GolombEncode(n, RBits) {
  q = n >> RBits;
  UnaryCode(q);
  WriteBits(n, RBits);
}
```

Output the lower (RBits) bits of n.

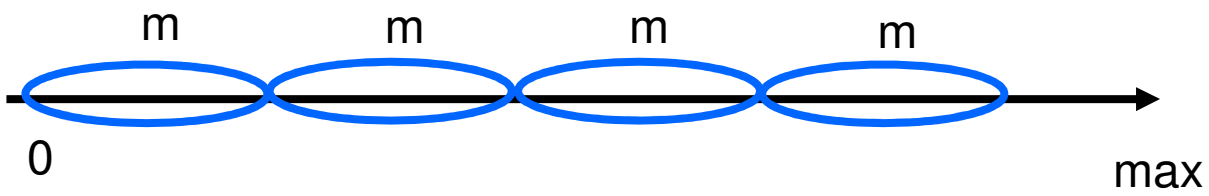
n	q	r	code
0	0	0	0000
1	0	1	0001
2	0	2	0010
3	0	3	0011
4	0	4	0100
5	0	5	0101
6	0	6	0110
7	0	7	0111

- Decoding:

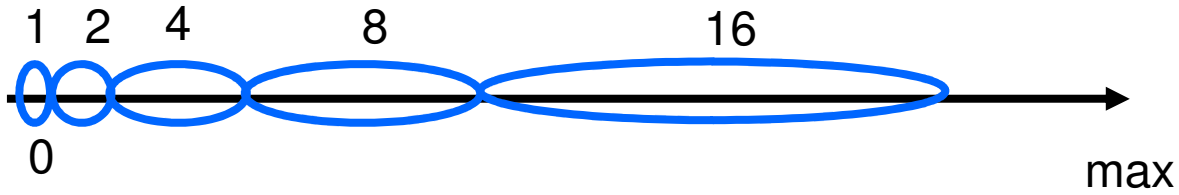
```
GolombDecode(RBits) {
  q = UnaryDecode();
  n = (q << RBits) + ReadBits(RBits);
  return n;
}
```

Exponential Golomb Code (Exp-Golomb)

- Golomb code divides the alphabet into groups of equal size



- In Exp-Golomb code, the group size increases exponentially
- Codes still contain two parts:
  - Unary code followed by fixed-length code



- Proposed by Teuhola in 1978

n	code
0	0
1	100
2	101
3	11000
4	11001
5	11010
6	11011
7	1110000
8	1110001
9	1110010
10	1110011
11	1110100
12	1110101
13	1110110
14	1110111
15	111100000

Implementation

- Encoding:
- Codeword for Group x > 0:
  - Unary code for x
  - X-bit index within the group
- Group ID: $\lfloor \log_2(n+1) \rfloor$

```
GetGroupID(n) {
  //Only for n > 0
  k = 2;
  While (n > (1 << k) - 2) {
    k++;
  }
  return k - 1;  //1, 2, 3...
}
```

2^1 - 2 →
2^2 - 2 →
2^3 - 2 →
2^4 - 2 ↘

n	code	ID
0	0	0
1	100	1
2	101	
3	11000	2
4	11001	
5	11010	
6	11011	
7	1110000	3
8	1110001	
9	1110010	
10	1110011	
11	1110100	
12	1110101	
13	1110110	
14	1110111	

Implementation

```
GetGroupID(n) {
  //Only for n > 0
  k = 2;
  While (n > (1 << k) - 2) {
    k++;
  }
  return k - 1;  //1, 2, 3...
}

ExpGolombEncode(n) {
  if (n == 0) {
    WriteBit(0);
  } else {
    GroupID = GetGroupID(n);
    UnaryEncode(GroupID);
    Index = n - ((1 << GroupID) - 1);
    WriteBits(Index, GroupID);
  }
}
```

n	code	Group ID
0	0	0
1	100	1
2	101	
3	11000	2
4	11001	
5	11010	
6	11011	
7	1110000	3
8	1110001	
9	1110010	
10	1110011	
11	1110100	
12	1110101	
13	1110110	
14	1110111	

Implementation

Decoding

```
ExpGolombDecode() {
    GroupID = UnaryDecode();
    if (GroupID == 0) {
        return 0;
    } else {
        Base = (1 << GroupID) - 1;
        Index = ReadBits(GroupID);
        return (Base + Index);
    }
  }
}
```

n	code	Group ID
0	0	0
1	100	1
2	101	
3	11000	2
4	11001	
5	11010	
6	11011	
7	1110000	3
8	1110001	
9	1110010	
10	1110011	
11	1110100	
12	1110101	
13	1110110	
14	1110111	

Why Golomb Code?

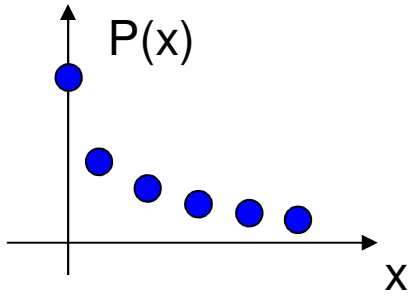
- Significance of Golomb code:
  - For any geometric distribution (GD), we can find a Golomb code that is optimal prefix code and is as close to the entropy as possible (among all prefix codes)
  - How to determine the Golomb parameter?
  - How to apply it into practical codec?

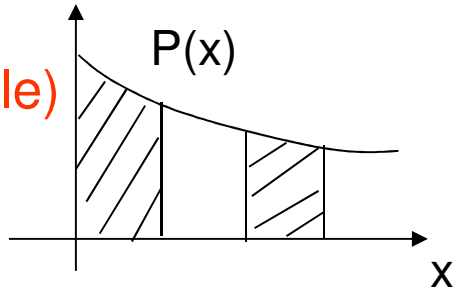
Optimal Code for Geometric Distribution

- Geometric distribution with parameter ρ:
  - $P(X=n) = \rho^n (1 - \rho)$
- Unary code is optimal prefix code when ρ ≤ 1/2.
  - Also optimal among all entropy coding for ρ = 1/2.
- How to make unary code optimal when ρ > 1/2 ?
  - Transform into GD with ρ ≤ 1/2 (as close as possible)

    How? By grouping m events together!

Each x can be written as $x = x_q m + x_r$





➔ x_q has geometric dist with parameter ρ^m.

Proof:

Optimal Code for Geometric Distribution

- How to encode the remainder Xr?
- For $P(X=n) = (1 - \rho)\,\rho^{n}$, if $\rho^{m} = 1/2$, then $P(X=n+m)=$

How to find the optimal Golomb parameter m? (In particular, $m = 2^{k}$).

Golomb Parameter Estimation (J2K book: pp. 55)

- Goal of adaptive Golomb code:
  - For the given data, find the best m such that $\rho^m \le 1/2$.
- How to find ρ from the statistics of past data?

$$P(x) = (1-\rho)\rho^x \quad \Rightarrow \quad E(x) = \sum_{x=0}^{\infty} (1-\rho)x\rho^x = (1-\rho)\frac{\rho}{(1-\rho)^2} = \boxed{\frac{\rho}{1-\rho}}$$

Method 1: $$k \ge \log_2\left(1/\log_2\left(\frac{1+E(x)}{E(x)}\right)\right).$$

Proof:

Golomb Parameter Estimation (J2K book: pp. 55)

$$E(x) = \frac{\rho}{1-\rho}$$

A faster method: Assume $\rho \approx 1$, $1 - \rho \approx 0$.

Adaptive Golomb Coding

- Initialize $A = A_0$, $N = 1$;
- For n = 0, 1, 2, …
  - k = max{0, ceil(log2(A / (2N)))};
  - Code symbol x_n using Golomb code with parameter k;
  - //Update after encoding!
  - If N = Nmax // Renormalize
    - A = floor(A / 2); N = floor(N / 2);
  - Else
    - $A = A + x_n$; N = N + 1;
  - End
- End

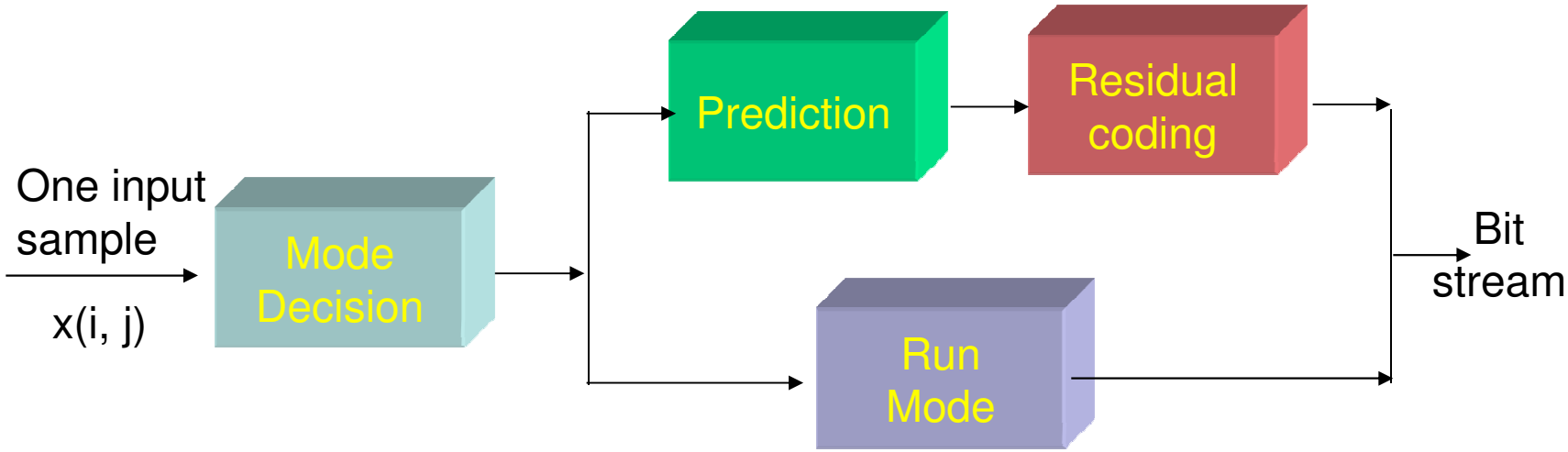
Purposes of Renormalization:

1. Prevent overflow.
2. Introduce a forgetting factor, so that the algorithm reacts to recent data more quickly.

Application in JPEG-LS

- JPEG-LS: Lossless JPEG
- The original JPEG also has a lossless algorithm, but has not been widely adopted.
- JPEG-LS:
  - Work began in 1995, completed in 1999.
  - Goals: Improved performance with minimal complexity.
  - Based on HP Lab's LOCO-I algorithm:
    - Low complexity compression of images
  - LOCO-I is a low complexity version of CALIC:
    - Context-based adaptive lossless image coding
  - JPEG-LS has better performance then lossless JPEG 2000.
- Techniques used:
  - Prediction
  - Adaptive Golomb code
  - Context adaptive coding
  - Run-length coding

JPEG-LS Block Diagram



- Code the input samples in raster scanning order.
- Run Mode:
  - Used in flat regions (when all 4 neighbors are equal), where prefix code is not efficient.
- Prediction Mode:
  - Used in other regions:
    - Predict from causal neighbors
    - Determine the context
    - In each context, find the probability model for the prediction residual.
    - Encode the residual with the adaptive Golomb code.

Prediction

c	b
a	x

- Predict each sample x from three neighbors:
  - Based on simple edge detection.

$$\hat{x} = \begin{cases} \min(a,b), & \text{if } c \geq \max(a,b) \\ \max(a,b) & \text{if } c \leq \min(a,b) \\ a+b-c, & \text{otherwise.} \end{cases}$$

or c = max(a, b, c)

or c = min(a, b, c)

Prediction residual: $\varepsilon = x - \hat{x}$

c = max(a, b, c)

c	b
a	x

$\hat{x} = \min(a,b) = b$

c	b
a	x

$\hat{x} = \min(a,b) = a$

c = min(a, b, c)

c	b
a	x

$\hat{x} = \max(a,b) = b$

c	b
a	x

$\hat{x} = \max(a,b) = a$

other

c	b
a	x

$\hat{x} = a + b - c$

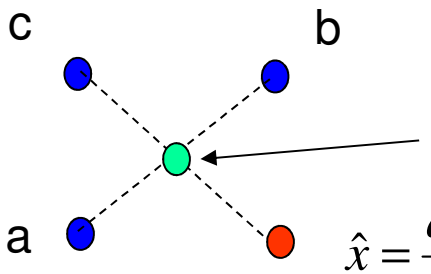
Prediction

other

c	b
a	x

$$\hat{x} = a + b - c$$

In this case, the predicted value is in the plane determined by a, b, and c.



$$\hat{x} = \frac{a+b}{2} + (\frac{a+b}{2} - c) = a + b - c.$$

- A equivalent representation of the JPEG-LS prediction rule:

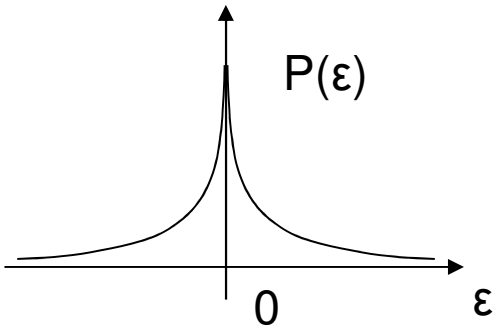
$$\hat{x} = median\{a,\ b,\ a+b-c\}.$$

Residual Distribution

- If the predictor is well designed, the residual can be modeled by two-sided geometric distribution:

$$p(\varepsilon) = \frac{1-\rho}{1+\rho}\rho^{|\varepsilon|}$$

$$0 < \rho < 1.$$



- Since we use integer predictor, a bias T is usually presented.
- The bias T can be divided into its integer part C and fractional part s:
  - T = C - s.
- C can be estimated and canceled:
  - Discussed later.
- So the remaining bias is

$$E\{\varepsilon\} = -s \qquad 0 \le s < 1.$$

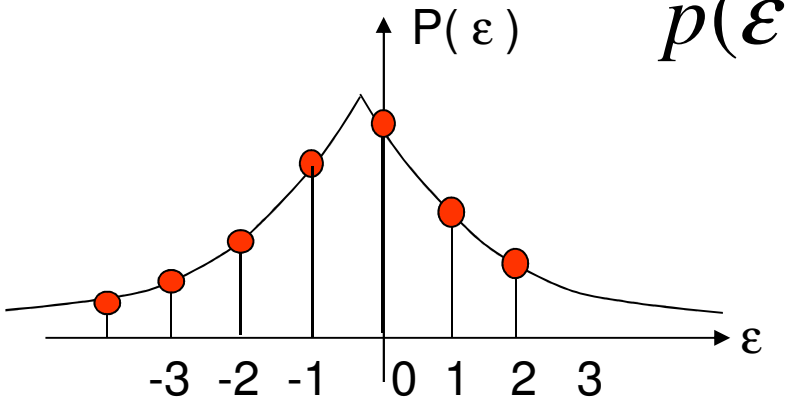
  - The value of S determines the optimal mapping from two-sided to one-sided GD.

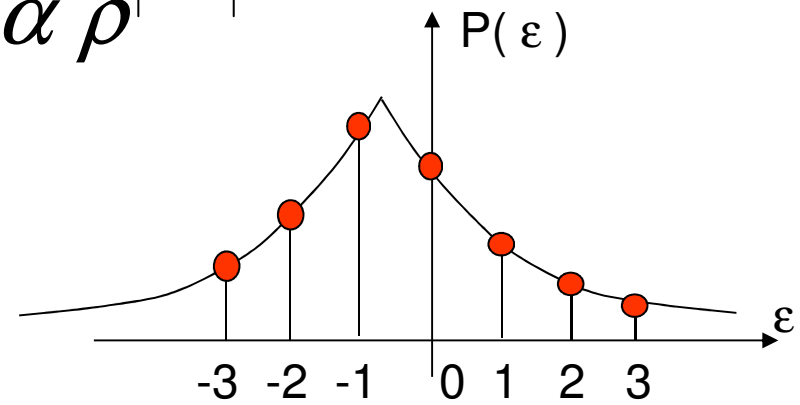
Residual Distribution: mapping to one-sided GD

Case 1: S ≤ 1/2

Case 2: S > 1/2

$$p(\varepsilon) = \alpha\, \rho^{|\varepsilon+s|}$$





If s ≤ 1/2:

P(0) ≥ P(-1) ≥ P(1) ≥ P(-2) ≥ …

$$\varepsilon' = 2|\varepsilon| - sign(\varepsilon)$$

ε	ε'
0	0
-1	1
1	2

If s > 1/2:

P(-1) ≥ P(0) ≥ P(-2) ≥ P(1) ≥ …

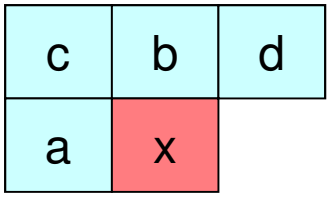
$$\varepsilon' = 2|\varepsilon+1| - sign(-(\varepsilon+1))$$

ε	ε'
-1	0
0	1
-2	2

$$sign(\varepsilon) = 1 \text{ if } \varepsilon < 0 \text{ and } 0 \text{ otherwise.}$$

$$\varepsilon' \approx 2|\varepsilon|$$

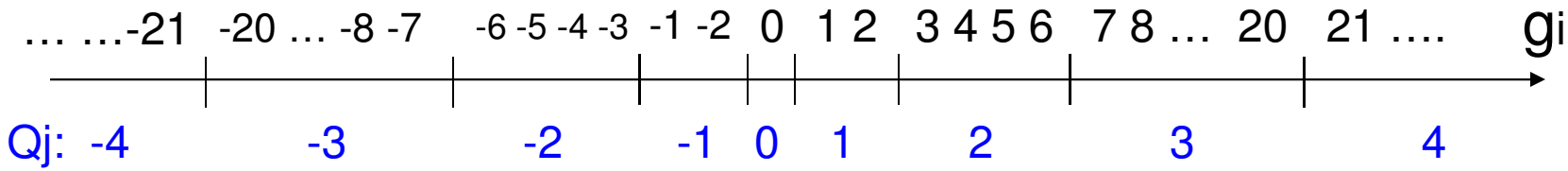
Context Model



- Each prediction residual is coded by adaptive Golomb code based on the local conditional probability: P(ε | neighbors)
- Context: Each configuration of the neighboring samples.
- Lower entropy (better compression) can be obtained by using higher order conditioning:
  - Recall: conditioning reduces entropy.
  - Problems:
    - Increased cost to keep track of more contexts.
    - Context dilution: don't have enough training data for each context.
- Contexts in JPEG-LS:
  - Based on 3 local gradients among 4 neighbors:
  - g1 = d – b,
  - g2 = b – c,
  - g3 = c – a.

Context Model

- g1 = d – b, g2 = b – c, g3 = c – a:
  - Too many {g1, g2, g3} possibilities if a, b, c, d all in [0, 255].
- Context Quantization:
  - Divide each gradient into roughly equiprobable regions
  - Aim: maximize the mutual information between x(n) and its context.
  - JPEG-LS: 9 regions for each gradient.



Total number of combinations of {g1, g2, g3}: 9 x 9 x 9 = 729.
Still too many !

Context Model

- Symmetry assumption:
  - To further reduce the number of contexts, It's assumed that

$$P\{\varepsilon = \Delta \mid \{g_1, g_2, g_3\}\} = P\{\varepsilon = -\Delta \mid \{-g_1, -g_2, -g_3\}\}$$

- In JPEG-LS, when the first nonzero gi is negative, we will use the context –{g1, g2, g3}, and encode ε as –ε.
- Decoder:

  When the decoder finds that the first nonzero gi is negative, it also uses –{g1, g2, g3} to decode the residual, then flip the sign.
- Final number of contexts: 365
- Proof:

Adaptive Golomb Code

- For each pixel:
  - 1. Find the prediction from neighbors.
  - 2. Decide the context index λ.
  - 3. Encode the prediction residual by adaptive Golomb code with the condition probability of the current context λ.
- To estimate the Golomb parameter:

Recall: $k = \max\left\{0, \quad \left\lceil \log_2\left(\frac{1}{2}E(x)\right)\right\rceil\right\}.$

- This should be applied to each context after mapping to 1-sided GD.

$$k = \max\left\{0, \quad \left\lceil \log_2\left(\frac{1}{2}E(\varepsilon'|\lambda)\right)\right\rceil\right\} \approx \max\left\{0, \quad \left\lceil \log_2\left(E(|\varepsilon| \ |\lambda)\right)\right\rceil\right\}$$

Recall: $\varepsilon' \approx 2|\varepsilon|$

Adaptive Golomb Code

$$m = 2^k \geq \frac{1}{2} E(x) \qquad k \approx \max\{0, \lceil \log_2 (E(|\varepsilon| \mid \lambda)) \rceil\}$$

- To facilitate the parameter estimation, for each context, we maintain:
  - A_λ: Sum of $|\varepsilon|$ in context λ.
  - N_λ: Total number of events in context λ.

$$k = \max\left\{0, \left\lceil \log_2 \frac{A_\lambda}{N_\lambda} \right\rceil\right\} \quad \text{or} \quad k = \min\left\{k' \middle| 2^{k'} N_\lambda \geq A_\lambda\right\}$$

C/C++ one-line implementation : for (k = 0; (N << k) < A; k++);

- To get rid of the integer bias C in prediction residual, we maintain in each context:
  - B_λ: Sum of all previous ε's in context λ.
  - ➔ $\frac{B_\lambda}{N_\lambda} \approx E\{\varepsilon \mid \lambda\}$ (-s in previous slides)

Adaptive Golomb Code

- Update after encoding:
  - $A_\lambda = A_\lambda + |\varepsilon|$;
  - $B_\lambda = B_\lambda + \varepsilon$;
  - If $N_\lambda = N_{max}$: $A_\lambda = A_\lambda / 2$; $B_\lambda = B_\lambda / 2$; $N_\lambda = N_\lambda / 2$;
  - $N_\lambda = N_\lambda + 1$.
- Bias update: Since integer bias was also removed in previous ε's, we don't expect too much value in B_λ/N_λ.
  - If $-1 \le B_\lambda/N_\lambda < 0$, stop. Otherwise:
    - If $B_\lambda/N_\lambda \le -1$:

      $C_\lambda = C_\lambda - 1$ //the minimum is -128

      $B_\lambda = B_\lambda + N_\lambda$ //Bring B_λ/N_λ back to [-1, 0]

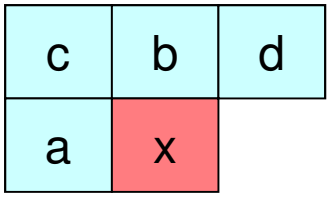
      if ($B_\lambda \le -N_\lambda$) $B_\lambda = -N_\lambda + 1$ //clamp B_λ/N_λ to -1.
    - If $B_\lambda / N_\lambda > 0$:

      $C_\lambda = C_\lambda + 1$ //the maximum is 127

      $B_\lambda = B_\lambda - N_\lambda$ //Bring B_λ/N_λ back to [-1, 0]

      if ($B_\lambda > 0$) $B_\lambda = 0$ //clamp to 0.

Run Mode



- The coder enters run mode when all four neighbors are identical:
  - X most likely has the same value
  - prefix code coding is not efficient.
  - Encode the run length **r**: the number of consecutive pixels with the same pixel value.

$$P(r) = (1-\rho)\rho^r$$

The run length r follows 1-sided geometric distribution.

Outline

- Unary Code
- Golomb Code
- Golomb-Rice Code
- Exponential Golomb Code
- Adaptive Golomb Code
  - Applications in JPEG-LS (Lossless JPEG)
  - Adaptive Run-Length/Golomb-Rice (RLGR) Code

Adaptive Run-Length/Golomb-Rice (RLGR) Code

- http://research.microsoft.com/pubs/102069/Malvar_DCC06.pdf
- Developed by Malvar at Microsoft Research
- A simple & adaptive combination of run-length coding and Golomb-Rice coding
- The GR code parameter k is adjusted by a backward adaptation rule
  - Less delay compared to forward adaptation
- Fractional adaptation is used.
- Nearly optimal for generalized Gaussian distributions

Main Encoding Rule of RLGR Code

- Has two main parameters: k and k_R

$k = 0$ "no run" mode	input symbol = u	code = $GR(u, k_R)$
$k > 0$ run mode	string of m symbols $u = 0$, with $m = 2^k$	code = 0
	string of m symbols $u = 0$ (with $m < 2^k$) followed by symbol $u \neq 0$	code = $1 + \text{bin}(m, k) + GR(u - 1, k_R)$

GR(u, k_R): The codeword of input u using a GR code with parameter k_R.
Bin(m, k): Binary representation of m using k bits.

Adaptation of k_R in RLGR Code

- Define $k_{RP} = L\, k_R$, where L is power of 2, e.g., L = 4.
- ➔ $k_R = k_{RP} >> \log_2(L)$.
- Let

$$p = \lfloor u / 2^{k_R} \rfloor = u >> k_R.$$

- After encoding an input by GR(u, k_R), k_R is updated as follows:

$p = 0$	decrease k_R by setting $k_{RP} \leftarrow k_{RP} - 2$
$p = 1$	no change in k_R
$p > 1$	increase k_R by setting $k_{RP} \leftarrow k_{RP} + p + 1$

- Intuition: if input is large, k_R should be increased to reduce the bits.
- Using k_{RP} instead of k_R leads to fractional adaptation.

Adaptation of k in RLGR Code

Define $k_P = L\,k$ ➔ $k = k_P >> \log_2(L)$.
(This also yields fractional adaptation)

$k = 0$	$u = 0 : k_P \leftarrow k_P + U_0$	
	$u > 0 : k_P \leftarrow k_P - D_0$	
$k > 0$	complete run	$k_P \leftarrow k_P + U_1$
	partial run	$k_P \leftarrow k_P - D_1$

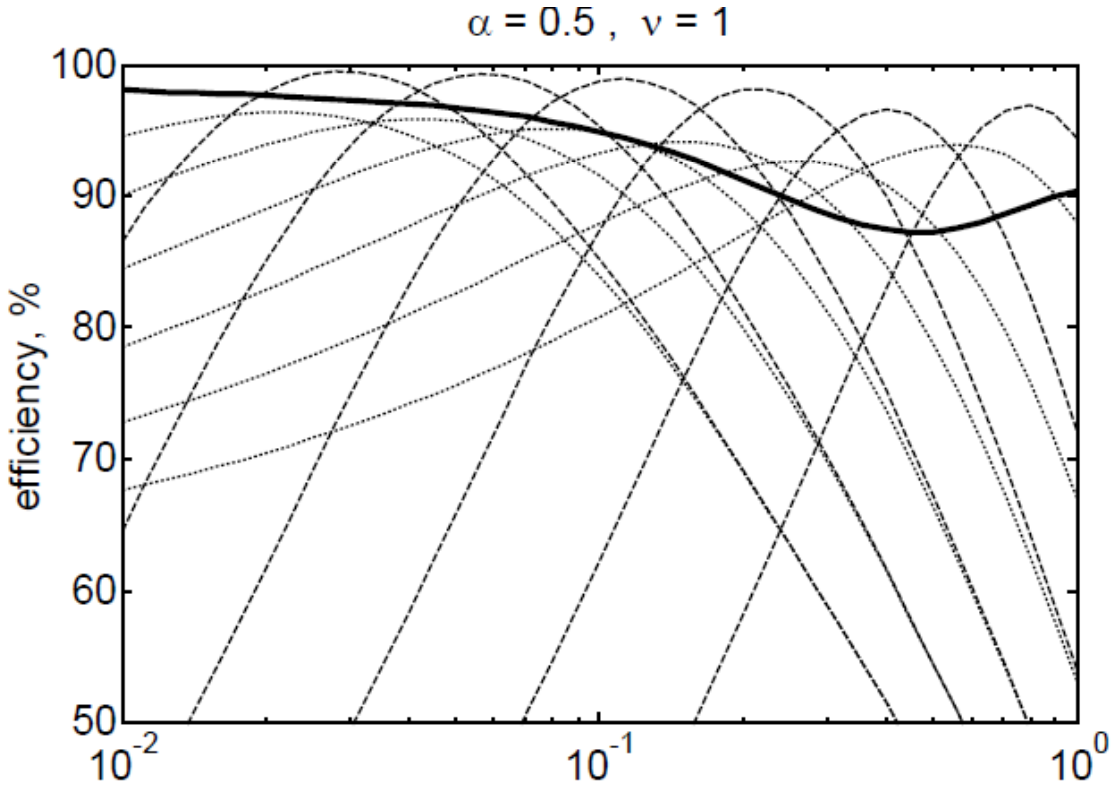
Table 4. Adaptation rule for the RLGR main parameter k, using fractional adaptation. It is applied immediately after the encoder outputs a full codeword; the new value of k is $\lfloor k_P / L \rfloor$.

The parameters {*U0, D0, U1, D1*} control the speed of adaptation;

Performance of RLGR Code

Comparison to GR and exponential Golomb code with different fixed parameters:

➔ RLGR nearly optimal, without knowing the input parameters.



Fast Learning Rate of RLGR Code

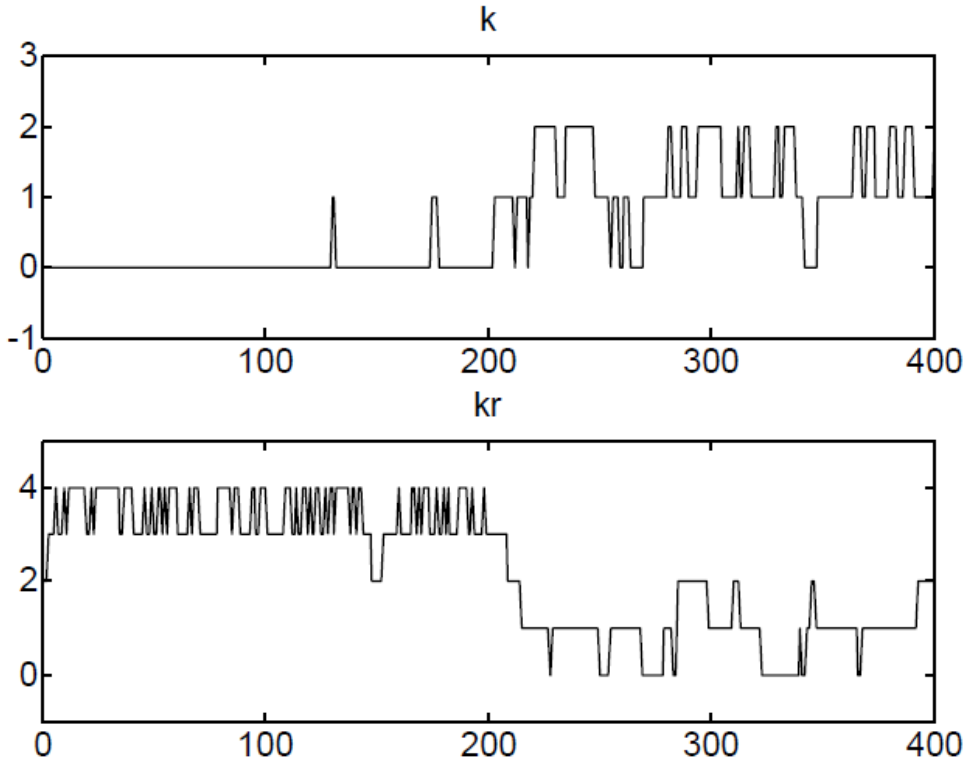


Figure 5. Fast learning rate for the RLGR coder. The source emits 400 symbols, and source and quantization parameters change at symbol 200. The RLGR parameters k and k_R quickly adapt to the new source statistics.

References

Summary of Golomb code and JPEG-LS:

- D. Taubman and M. Marcellin, JPEG2000 image compression fundamentals, standards and practice, Kluwer Academic, Boston, 2002.

Golomb-Rice Code:

- S. Golomb, Run-length encodings, IEEE Trans. Information Theory, Vol. 12, No. 3, Jul 1966, pp. 399 - 401.
- R. Gallager, D. van Voorhis, Optimal source codes for geometrically distributed integer alphabets, IEEE Trans. Information Theory, Vol. 21, No. 2, Mar 1975, pp. 228 – 230.

JPEG-LS:

- M. Weinberger, G. Seroussi and G. Sapiro, The *LOCO-I lossless image compression algorithm: principles and standardizations into JPEG-LS, IEEE Trans. Image Processing, Vo.. 9(8), pp. 1309-1324, Aug. 2000.*
- N. Memon, *Adaptive coding of DCT coefficients by Golomb-Rice codes*, HP Lab Technical Report, HPL-98-146.
- X. Wu and N. Memon, Context-based, adaptive, lossless image coding, IEEE Trans. Communications, 45(4), pp. 437-444, Apr. 1997.

RLGR:

- H. S. Malvar, Adaptive run-length/Golomb-Rice encoding of quantized generalized Gaussian Sources with unknown statistics, DCC'06.