Data Compression

Arithmetic Coding

Chang-Su Kim

Arithmetic Coding

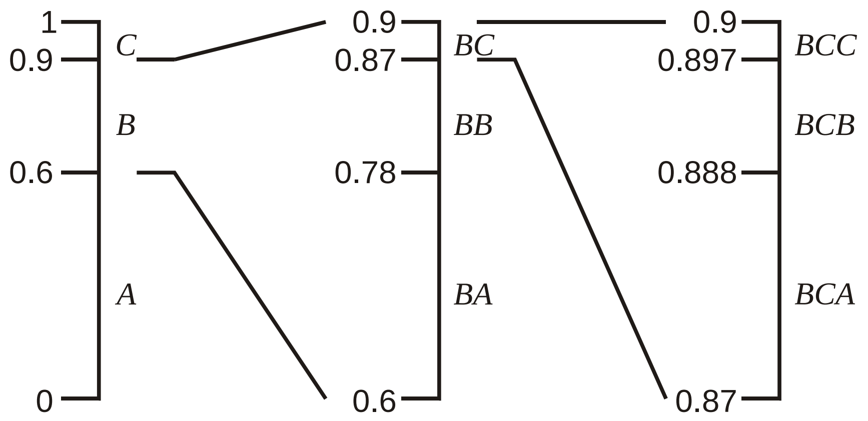
- Huffman coding is not always the best option
  - It is optimum only among the coding schemes, which assign a fixed, integer number of bits to each symbol.
  - For two symbol alphabet, it always assigns 0 to one symbol and 1 to the other symbol
    - Average length = 1 bit/symbol
    - $p_0 = 0.9999$, $p_1 = 0.0001 \rightarrow$ entropy = 0.00147 bit/symbol

- Arithmetic coding is better than Huffman coding
  - Coding efficiency
  - Adaptivity
  - Recent video coding standards incorporates arithmetic coding

Example 1

- Alphabet = {A, B, C}
- p(A) = 0.6, p(B) = 0.3, p(C) = 0.1

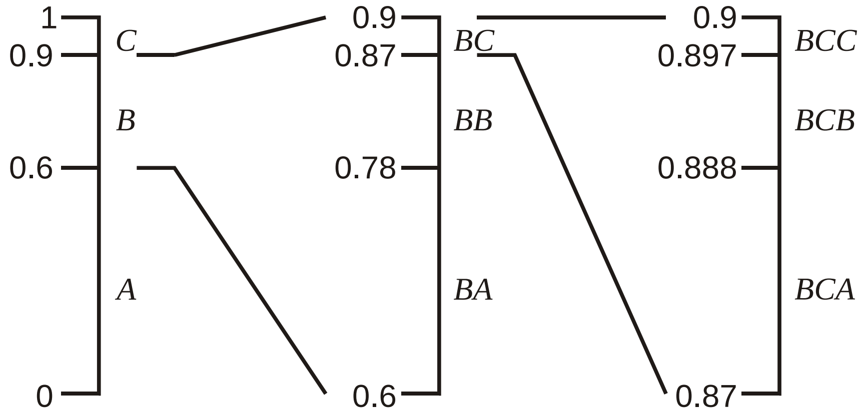
- The messages starting with A, B and C are respectively mapped to the half-open intervals [0, 0.6), [0.6, 0.9), and [0.9, 1), according to their probabilities



Example 1

- Alphabet = {A, B, C}
- p(A) = 0.6, p(B) = 0.3, p(C) = 0.1

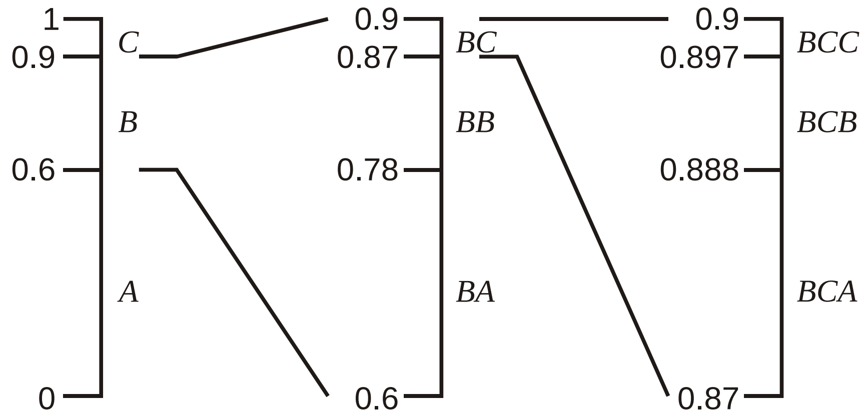
- Let us encode a message BCA. Since it starts with B it is first mapped to the interval [0.6, 0.9).



Example 1

- Alphabet = {A, B, C}
- p(A) = 0.6, p(B) = 0.3, p(C) = 0.1

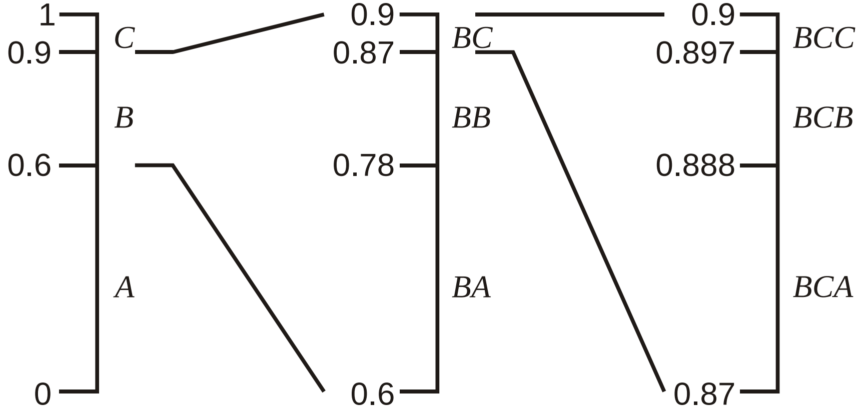
- The interval is then divided into three subintervals [0.6, 0.78), [0.78, 0.87), [0.87, 0.9), corresponding to the messages starting with BA, BB, BC, respectively. Note that the length ratio of the subintervals is set to
  - 0.18 : 0.09 : 0.03 = 6 : 3 : 1 = p(A) : p(B) : p(C)



Example 1

- Alphabet = {A, B, C}
- p(A) = 0.6, p(B) = 0.3, p(C) = 0.1

- Similarly, [0.87, 0.9) is further divided into three subintervals, and the messages starting with BCA are mapped to [0.87, 0.888).

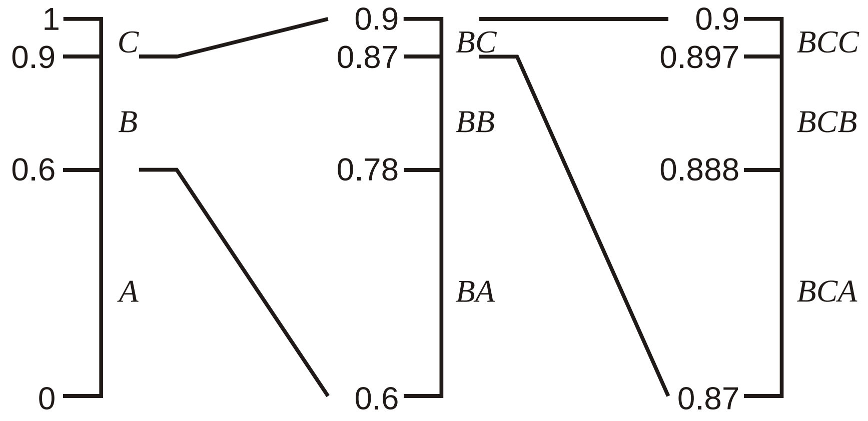


Example 1

- Alphabet = {A, B, C}
- p(A) = 0.6, p(B) = 0.3, p(C) = 0.1

- The two end points can be written in binary numbers as

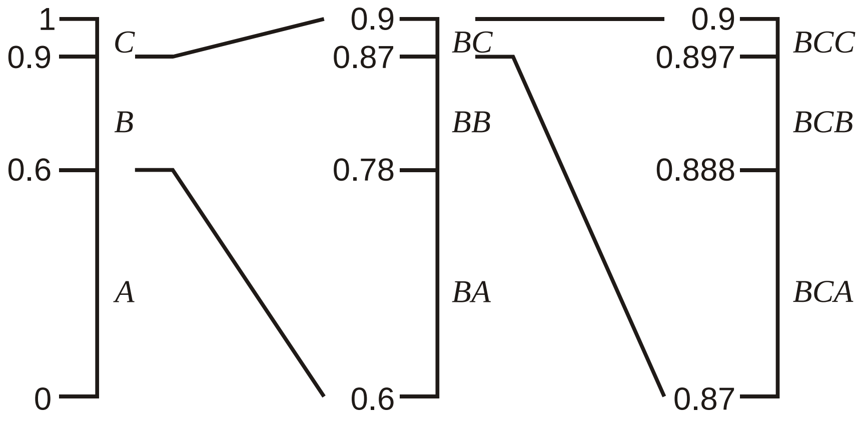
$$0.87 = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^4} + \frac{1}{2^5} + \cdots = 0.11011\cdots,$$
$$0.888 = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^7} + \cdots = 0.11100\cdots.$$



Example 1

- Alphabet = {A, B, C}
- p(A) = 0.6, p(B) = 0.3, p(C) = 0.1

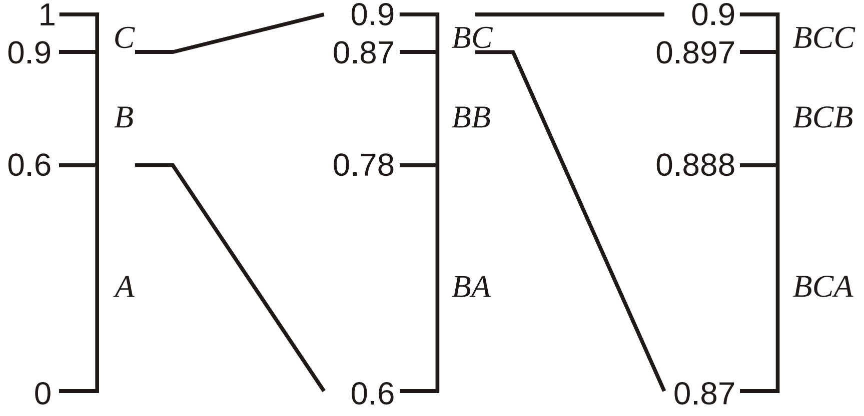
- The encoder can transmit an arbitrary number within this interval to specify that the message starts with BCA. For example, it can transmit the sequence of three bits, 111, which corresponds to 0.875 in decimal.



Example 1

- Alphabet = {A, B, C}
- p(A) = 0.6, p(B) = 0.3, p(C) = 0.1

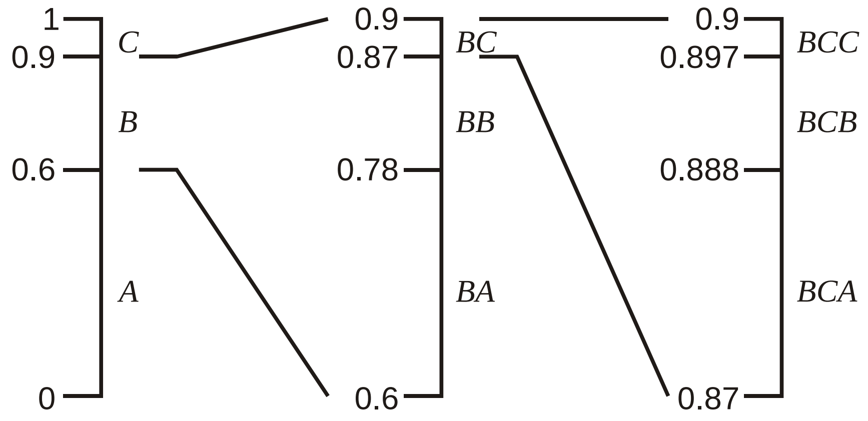
- Given 111, the decoder can follow the same division procedure, and know that 0.875 lies within the interval [0.87, 0.888), hence the message starts with BCA.



Example 1

- Alphabet = {A, B, C}
- p(A) = 0.6, p(B) = 0.3, p(C) = 0.1

- However, 0.875 can represent B, BA, or BAC. One way to resolve this issue is to use one more symbol EOS (end of sequence).



Example 2

Example 4.3.1:

Consider a three-letter alphabet $\mathcal{A} = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, and $P(a_3) = 0.2$. Using the mapping of Equation (4.1), $F_X(1) = 0.7$, $F_X(2) = 0.8$, and $F_X(3) = 1$. This partitions the unit interval as shown in Figure 4.1.

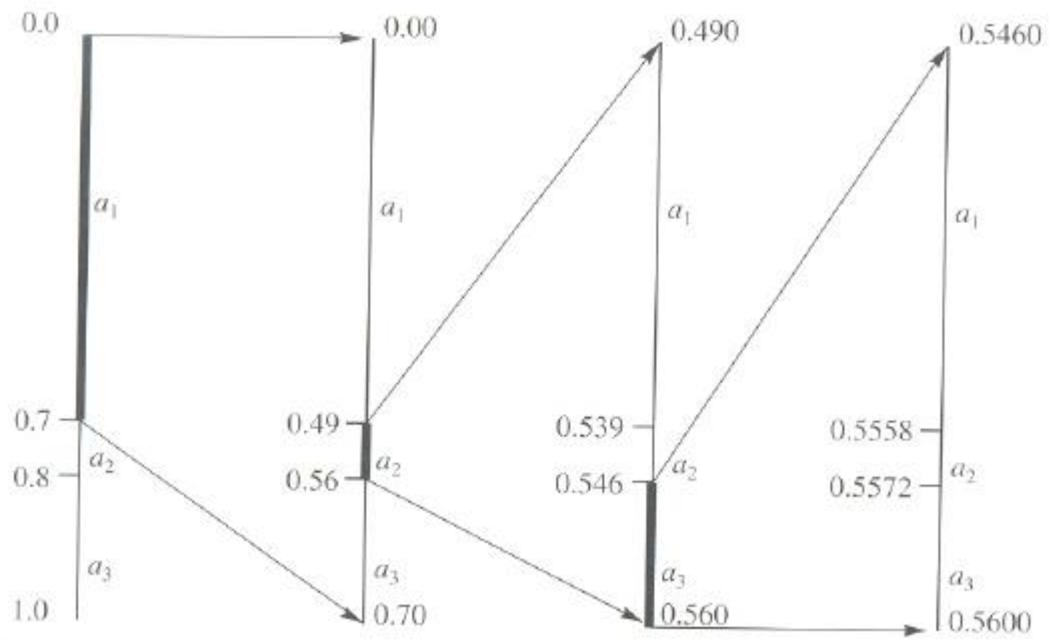


FIGURE 4.1 Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.

Generating A Tag

- We assume that the alphabet = {*1,2,3,…,m*}
- Cumulative distribution function

$$F(i) = \sum_{k=1}^{i} P(k)$$

- The symbol *i* represented by [*F*(*i*-1), *F*(*i*)).
- A tag denotes a number in the interval, so it uniquely represents the symbol. We will use the midpoint

$$T(i) = F(i-1) + \frac{1}{2} P(i)$$

Generating A Tag

- We are encoding a sequence of symbols $\mathbf{x}^{(n)} = (x_1, x_2, \cdots, x_n)$
- As we encode more symbols, the interval gets smaller.

1) $[l^{(n)}, u^{(n)})$: the interval for $\mathbf{x}^{(n)}$

2) Initial interval and tag

$$l^{(1)} = F(x_1 - 1)$$

$$u^{(1)} = F(x_1)$$

$$T^{(1)} = \frac{l^{(1)} + u^{(1)}}{2}$$

Interval length = $u^{(1)} - l^{(1)} = P(x_1)$

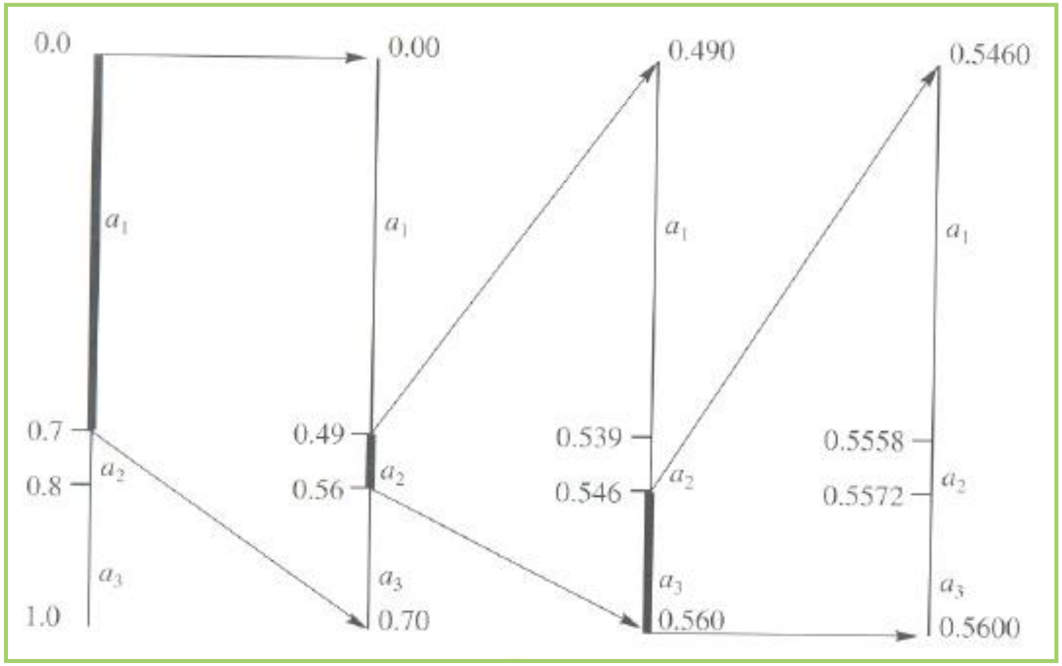
3) Iteration

$$l^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)}) F(x_n - 1)$$

$$u^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)}) F(x_n)$$

$$T^{(n)} = \frac{l^{(n)} + u^{(n)}}{2}$$

Interval length = $u^{(n)} - l^{(n)} = (u^{(n-1)} - l^{(n-1)}) P(x_n) = P(\mathbf{x}^{(n)})$



Generating A Binary Code

1) The tag $T^{(n)}$ is inside the interval $[l^{(n)}, u^{(n)})$, so it uniquely represents the sequence $\mathbf{x}^{(n)}$.
2) Given the tag, we can decode the sequence $\mathbf{x}^{(n)}$.
3) However, we need to express the tag with a finte number of bits.
4) How can we do achieve this?

By truncating $T^{(n)}$ to $\tilde{T}^{(n)}$ with $\left\lceil \log \frac{1}{P(\mathbf{x}^{(n)})} \right\rceil + 1$ bits, we can assure that

$\tilde{T}^{(n)}$ is still in the interval.

Incremental Coding: Brief Explanation

- Encoding
  - If the interval is entirely within [0, 0.5), put 0
  - If the interval is entirely within [0.5, 1), put 1
  - And so forth.

- Decoding
  - Given the series of bits (b0, b1, b2, …) , if the first k bits unambiguously represent a number in $[l^{(1)},u^{(1)})$, output the first symbol.
  - And so forth

- Because we are dealing with the numbers with “int” type, which uses only four bytes, the implementation is much more complex.

Application: Binary Image Coding

- Each pixel is binary, which represents 1 (black) or 0 (white)

Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images

STUART GEMAN AND DONALD GEMAN

Abstract—We make an analogy between images and statistical mechanics systems. Pixel gray levels and the presence and orientation of edges are viewed as states of atoms or molecules in a lattice-like physical system. The assignment of an energy function in the physical system determines its Gibbs distribution. Because of the Gibbs distribution, Markov random field (MRF) equivalence, this assignment also determines an MRF image model. The energy function is a more convenient and natural mechanism for embodying picture attributes than are the local characteristics of the MRF. For a range of degradation mechanisms, including blurring, nonlinear deformations, and multiplicative or additive noise, the posterior distribution is an MRF with a structure akin to the image model. By the analogy, the posterior distribution defines another (imaginary) physical system. Gradual temperature reduction in the physical system isolates low energy states ("annealing"), or what is the same thing, the most probable states under the Gibbs distribution. The analogous operation under the posterior distribution yields the maximum *a posteriori* (MAP) estimate of the image given the degraded observations. The result is a highly parallel "relaxation" algorithm for MAP estimation. We establish convergence properties of the algorithm and we experiment with some simple pictures, for which good restorations are obtained at low signal-to-noise ratios.

Index Terms—Annealing, Gibbs distribution, image restoration, line process, MAP estimate, Markov random field, relaxation, scene modeling, spatial degradation.

I. Introduction

THE restoration of degraded images is a branch of digital

The essence of our approach to restoration is a stochastic relaxation algorithm which generates a sequence of images that converges in an appropriate sense to the MAP estimate. This sequence evolves by *local* (and potentially *parallel*) changes in pixel gray levels and in locations and orientations of boundary elements. Deterministic, iterative-improvement methods generate a sequence of images that monotonically increase the posterior distribution (our "objective function"). In contrast, stochastic relaxation permits changes that *decrease* the posterior distribution as well. These are made on a *random* basis, the effect of which is to avoid convergence to *local maxima*. This should not be confused with "probabilistic relaxation" ("relaxation labeling"), which is deterministic; see Section X.

The stochastic relaxation algorithm can be informally described as follows.

1) A local change is made in the image based upon the current values of pixels and boundary elements in the immediate "neighborhood." This change is *random*, and is generated by sampling from a local conditional probability distribution.

2) The local conditional distributions are dependent on a global control parameter T called "temperature." At *low* temperatures the local conditional distributions concentrate on states that *increase* the objective function, whereas at high temperatures the distribution is essentially uniform. The limiting cases $T=0$ and $T=\infty$ correspond respectively to greedy

Arithmetic Coding Revis

ALISTAIR MOFFAT
The University of Melbourne
RADFORD M. NEAL
University of Toronto

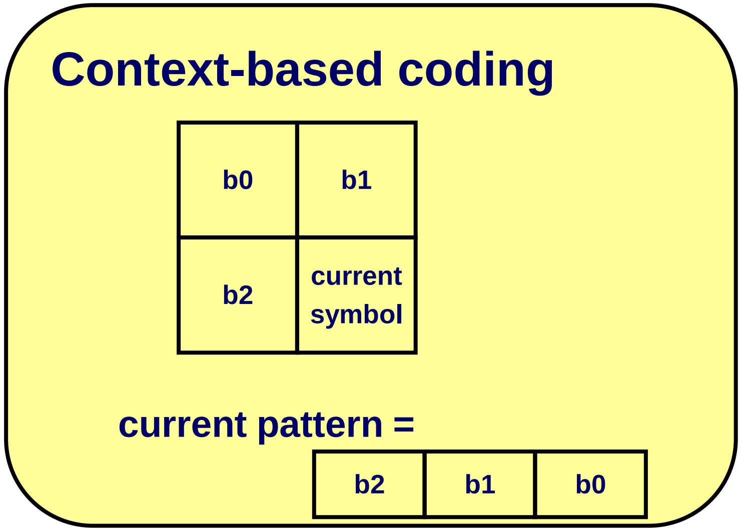
Adaptation Using Single Contexts

```
BinImage Image; // binary image similar to CharImage class
Image.Load("original.bim");

COutStream Out;
Out.open("original.cmp");
/* Header for image size */
Out.putvlc(Image.WX, 16);
Out.putvlc(Image.WY, 16);

// Arithmetic Encoding start
start_arithmetic_encode();   // initialization
context *pContext = create_context_easy(2);

for(dy=0; dy<Image.WY; dy++)
  for(dx=0; dx<Image.WX; dx++){ // raster scan order
    int current_symbol = Image.GetPixel(dx, dy);
    encode(pContext, current_symbol, Out);
  }

delete_context(pContext); // free memory for context
finish_arithmetic_encode(Out); // arithmetic coding termination
Out.close();
```

Adaptivity

input		0	0	1	1	0	0	0	1
p(0)	1/2	2/3	3/4	3/5	3/6	4/7	5/8	6/9	6/10
p(1)	1/2	1/3	1/4	2/5	3/6	3/7	3/8	3/9	4/10

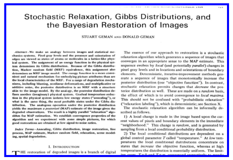
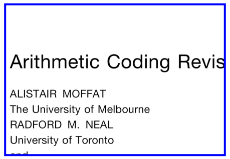
Adaptation Using Multiple Contexts




```
start_arithmetic_encode();
context *pContext[8];
for(p=0; p<8; p++)
  pContext[p] = create_context_easy(2);

for(dy=0; dy<Image.WY; dy++)
  for(dx=0; dx<Image.WX; dx++){
    int current_symbol = Image.GetPixel(dx, dy);

    int current_pattern = Image.GetPixel(dx-1, dy); // left pixel
    current_pattern = (current_pattern << 1) + Image.GetPixel(dx, dy-1); // upper pixel
    current_pattern = (current_pattern << 1) + Image.GetPixel(dx-1, dy-1); // upper left pixel

    encode(pContext[current_pattern], current_symbol, Out);
  }

for(p=0; p<8; p++)
  delete_context(pContext[p]);
finish_arithmetic_encode(Out);
Out.close();
```

Results

File size (bytes) [compression ratio]

	Original	Single Context	Multiple Contexts
 Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images STUART GEMAN AND DONALD GEMAN	100,004	47,211 [2.12]	21,621 [4.62]
 Arithmetic Coding Revis ALISTAIR MOFFAT The University of Melbourne RADFORD M. NEAL University of Toronto	100,004	30,476 [3.28]	3,276 [30.53]

Arithmetic Coder

- Core files
  - arith.cpp
  - context.cpp
  - bitio.cpp
  - Note: If you use these files for research, please give a reference to the paper

    A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic Coding Revisited," *ACM Trans. Information Systems*, vol. 16, no. 3, pp.256–294, July 1998
- Usage
  - The last commented part of “example1.cpp”
- Additional files for binary image coding
  - CharImage.cpp
  - BinImage.cpp