Christopher Tsai
January 25, 2008
EE 398 – Image Compression

Problem Set II – Arithmetic Coding

Problem #1 – Geometric Source

Probability Mass Function of a White Run: $p_{L_W}(l_W) = (1-p_{WB})^{l_W-1}p_{WB}$

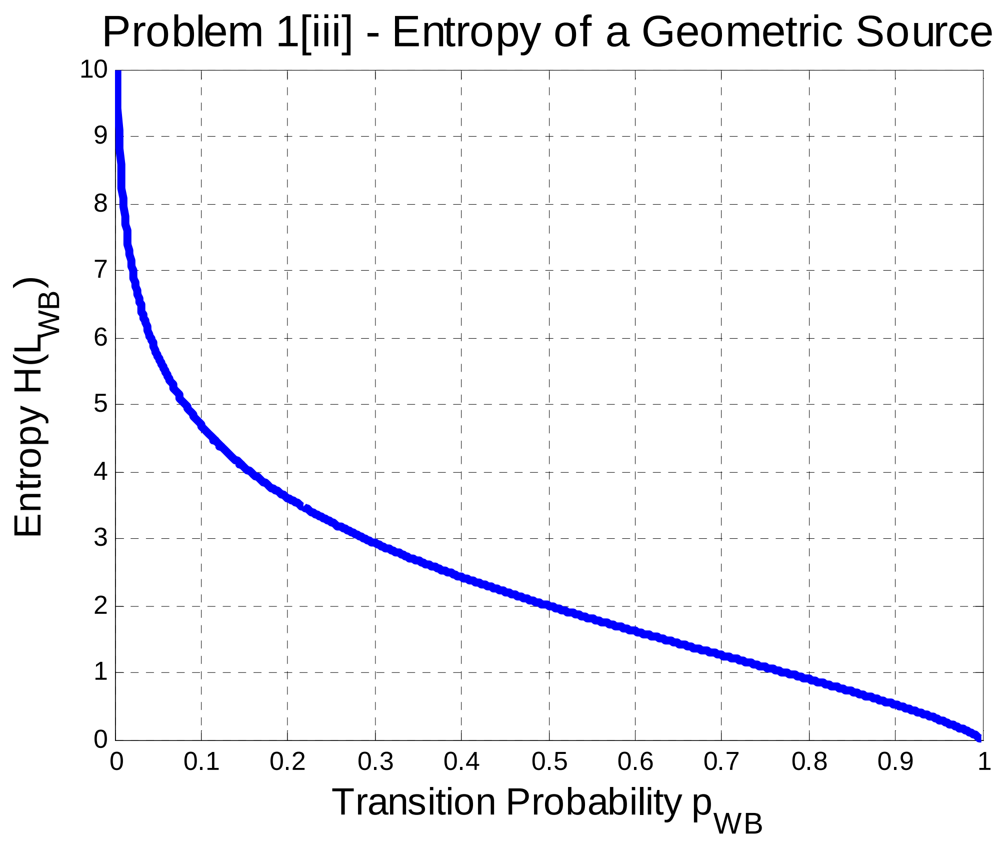
Probability Mass Function of a Black Run: $p_{L_B}(l_B) = (1-p_{BW})^{l_B-1}p_{BW}$

Expected Length of a White Run: $E[L_W] = \frac{1}{p_{WB}}$

Expected Length of a Black Run: $E[L_B] = \frac{1}{p_{BW}}$

Entropy of White Sequence Length: $H(L_W) = \frac{1-p_{WB}}{p_{WB}}\mathbf{log}_2\left(\frac{1}{1-p_{WB}}\right) + \mathbf{log}_2\left(\frac{1}{p_{WB}}\right)$

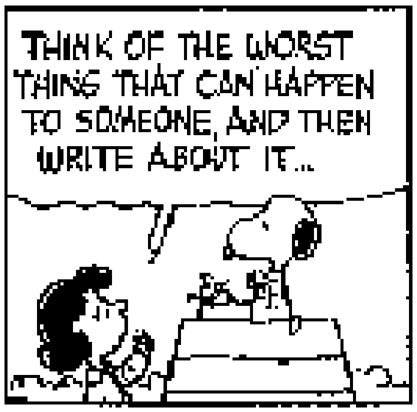
Entropy of Black Sequence Length: $H(L_B) = \frac{1-p_{BW}}{p_{BW}}\mathbf{log}_2\left(\frac{1}{1-p_{BW}}\right) + \mathbf{log}_2\left(\frac{1}{p_{BW}}\right)$



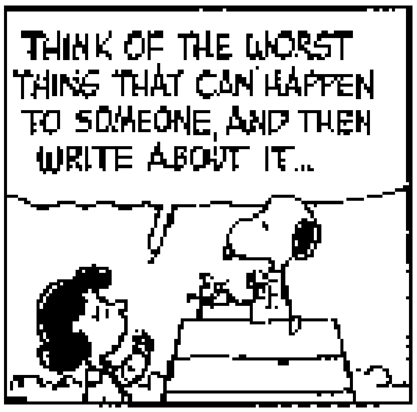
Problem #2 – Binary Arithmetic Coder

By comparing the number of zeros in the vectorized *Snoopy* image to the number of ones, we estimate the probability $f_X(x = 0) \approx \boxed{\mathbf{0.2121582}}$, or approximately 21%. Arithmetically encoding this vectorized image stream, we achieve a $\boxed{\textbf{compression ratio of approximately 1.3380155}}$. The pre-encoded image stream comprises 16384 bits, whereas our encoded sequence consumes only 12245 bits. The reconstructed image is a perfect reproduction of the original, pre-encoded image, which we expect from a lossless algorithm such as the arithmetic coding routine we implemented:

Original Snoopy


THINK OF THE WORST THING THAT CAN HAPPEN TO SOMEONE, AND THEN WRITE ABOUT IT...

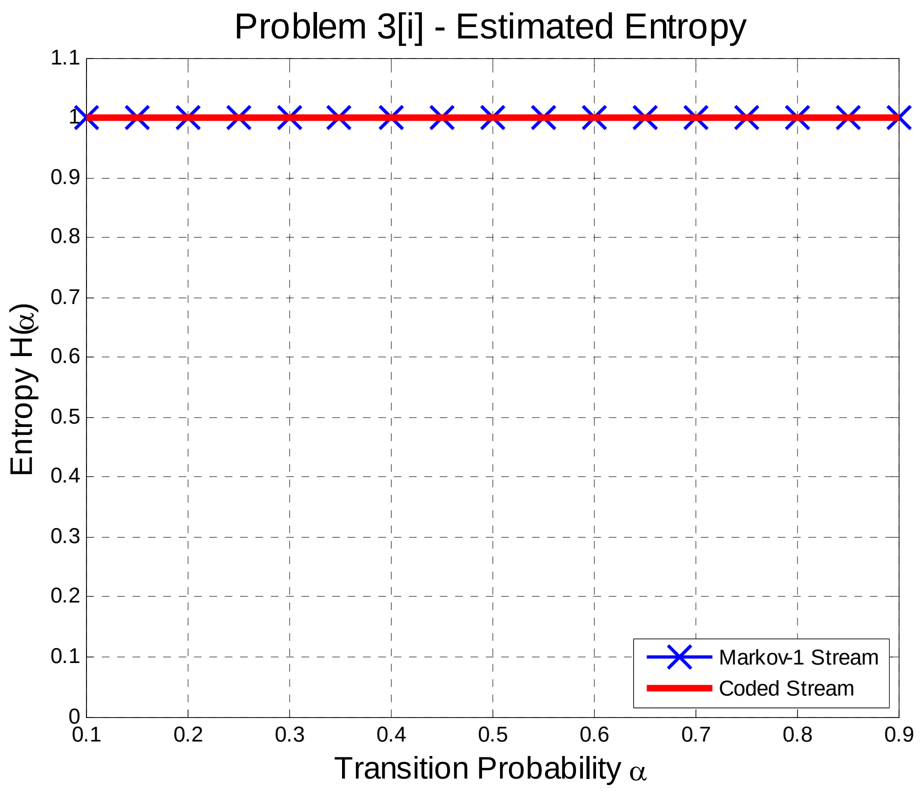
Reconstructed Snoopy


THINK OF THE WORST THING THAT CAN HAPPEN TO SOMEONE, AND THEN WRITE ABOUT IT...

Problem #3 – Arithmetic Coding of Markov-1 Source

For a Markov-1 source with symmetric transition probabilities, the number of ones will closely parallel the number of zeros; thus, with an approximately equal balance of ones and zeros, the estimated entropy of both the original source and the coded sequence hovers about

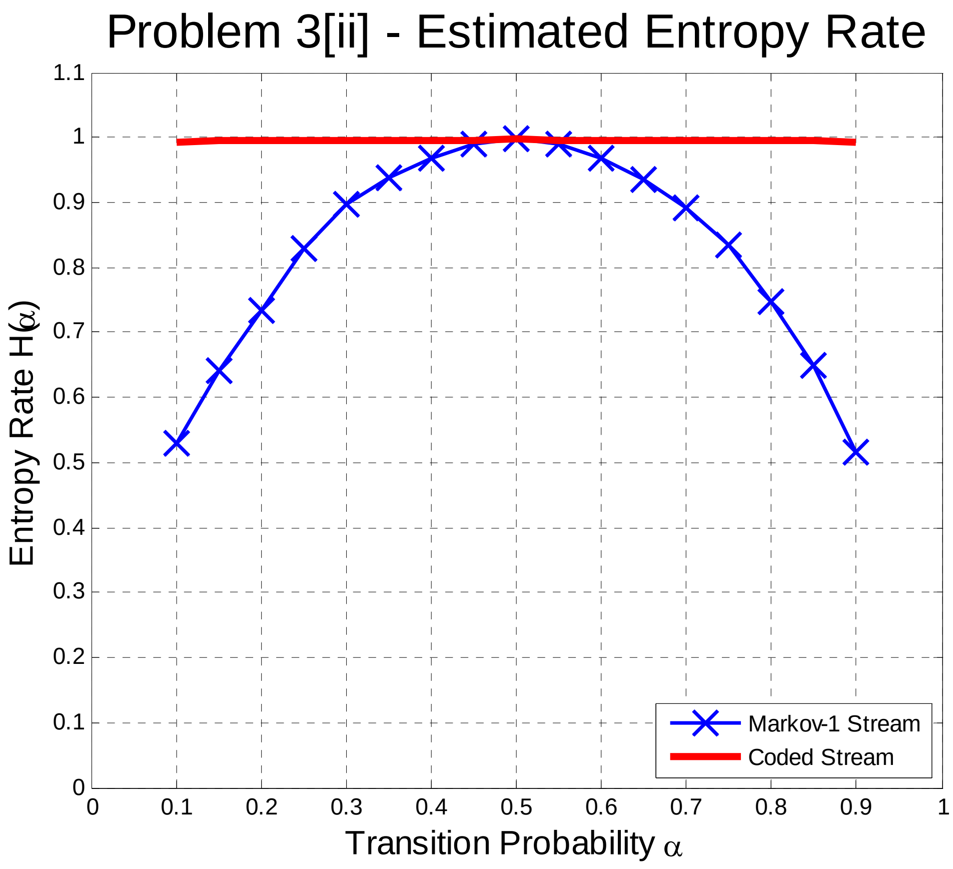
$$H(\alpha) = \frac{1}{2}\log_2 2 + \frac{1}{2}\log_2 2 = 1$$



The inherent entropy rate of the Markov-1 source is *not* uniformly unity, because symbols generally depend closely on their previous values. Only for the source with symmetric transition probabilities (α = 0.5) is the entropy rate approximately unity. Generally, uneven transition probabilities lead to state dependence, which, as the plot reveals, manifests itself in the sub-unity entropy rate. We remedy this sub-optimal entropy rate through arithmetic coding, which allows us to approach the ideal entropy rate; to adapt our arithmetic coder to a Markov source, we introduce *two* probability statistics in our symbol probabilities, depending on the current state of the input:

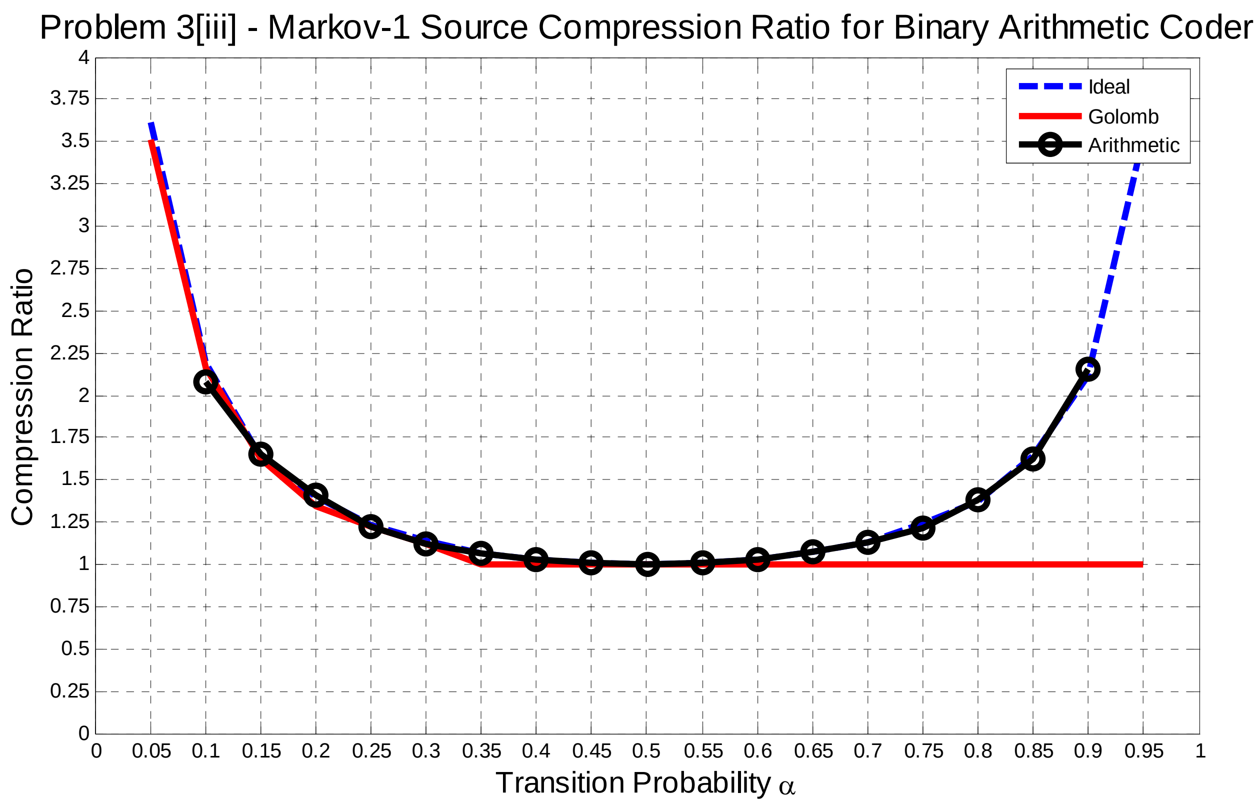
```
% Markov-1 source characteristics:
symbol1 = data(1);
p00 = floor(prob00 * 2^P);      % prob00 = prob{0 -> 0} = 1 - alpha
p10 = floor(prob10 * 2^P);      % prob10 = prob{1 -> 0} = beta

                                        ⋮

for n = 2 : inLength

    if (data(n-1) == '0')
        T = A*p00;
    else
        T = A*p10;
    end

                                        ⋮
```

Once adapted to accommodate Markov-1 source state differences, our binary arithmetic coder strives to bring the entropy rate to unity, so that encoded symbols have as little statistical dependence as possible:



Indeed, the encoded symbols are now virtually independent, with equal numbers of zeros and ones.

The compression now resembles the ideal entropy code compression that we computed analytically in Problem Set #1. The arithmetic encoder, by bringing the entropy rate as close as possible to unity without compromising image data, effectively removes all statistical redundancy and achieves the ideal limit of lossless compression. As we observe in the following graph, the Golomb code remains competitive up to $\alpha = 0.5$, at which point it departs from the ideal compression ratio:



Even for $\alpha < 0.5$ – relatively stable Markov processes – the Golomb code performs slightly below the arithmetic code, suggesting that, despite its optimality for geometric sources, the Golomb code cannot remove all redundancy from Markov sources. The arithmetic code, though complicated to generate, removes redundancy much better, its performance curve corroborates.