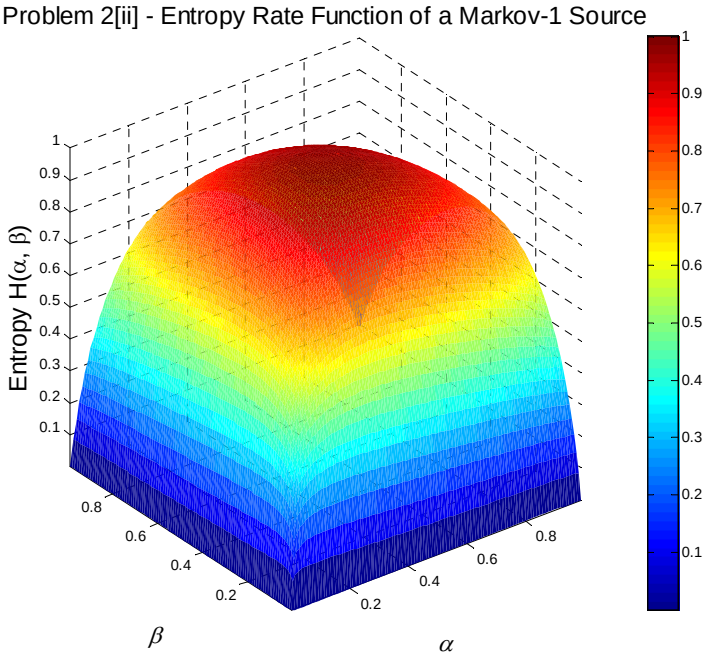
Christopher Tsai
January 20, 2008
EE 398 – Image Compression

Problem Set I – Lossless Coding

Problem #2 – Markov-1 Source Model

The two-dimensional entropy rate function $H(\alpha, \beta)$ is

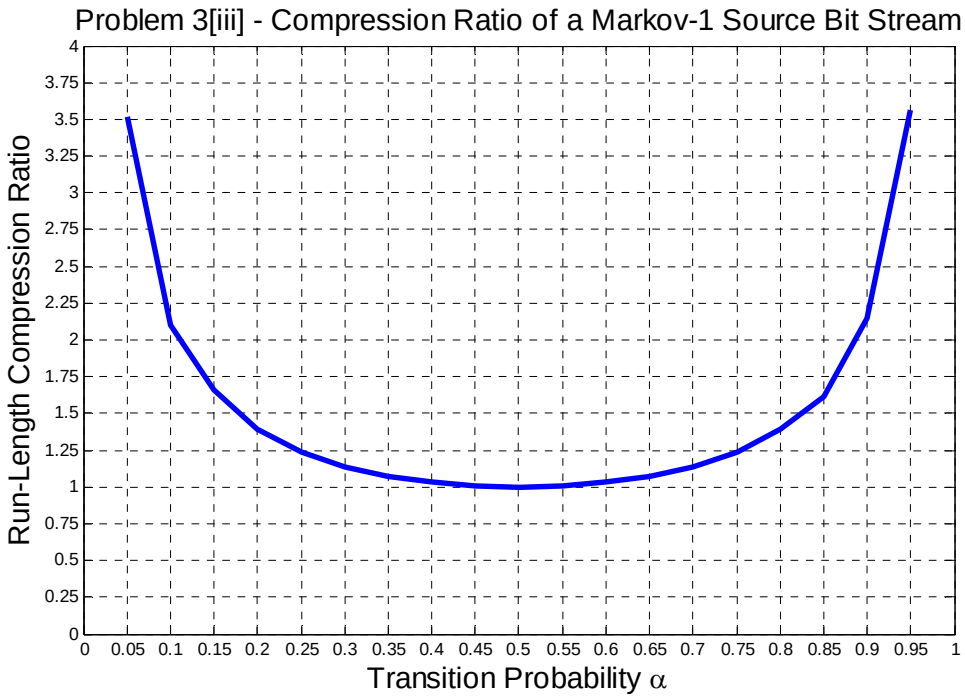
$$H(\alpha,\beta) = -\frac{\beta}{\alpha+\beta}[(1-\alpha)\,\mathbf{log}_2(1-\alpha) + \alpha\,\mathbf{log}_2\,\alpha] - \frac{\alpha}{\alpha+\beta}[(1-\beta)\,\mathbf{log}_2(1-\beta) + \beta\,\mathbf{log}_2\,\beta]$$



For both special cases, the entropy rate reduces to $H(\{X_n\}) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)$. For the case $\alpha = \beta$, the transition probabilities are identical, leading to an equal probability and proportion of ones and zeros. In other words, such a source is state-independent; every transition matrix looks identical. For the case $\alpha + \beta = 1$, the two transition probabilities are complements; any given bit is zero with probability β, and one with probability α. In this case, α directly denotes the proportion of ones in the sequence, whereas β = 1 – α approximates the number of zeros. For $\alpha = \beta = 0.2$, our test sequence has a tenth-order entropy of approximately 0.725238 bit, which closely approximates the analytical entropy rate of 0.721928 bit.

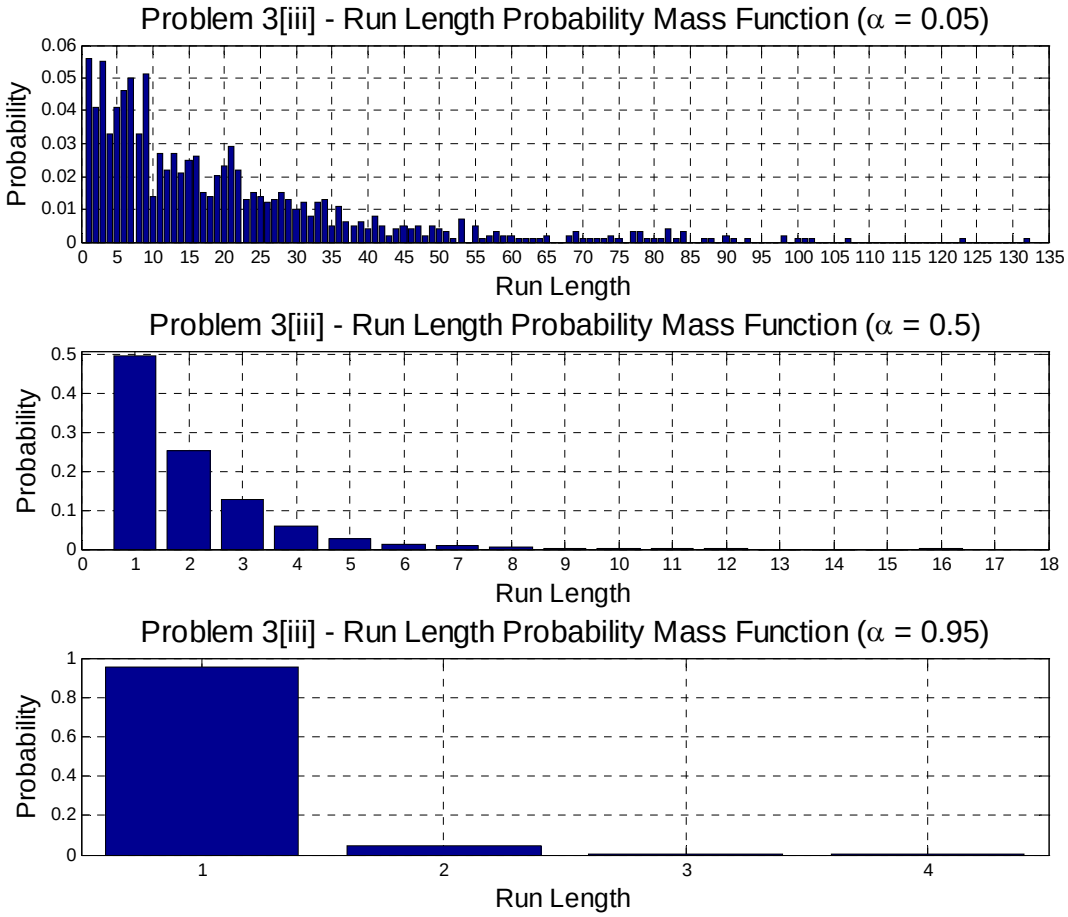
Problem #3 – Run-Length Coding

If we compress a Markov-1 source sequence with run-length encoding, we can achieve superior compression ratios for low $\alpha = \beta$ and for high $\alpha = \beta$ than we can for intermediate values.



This plot reveals that, when the probability α is low (less than 0.1), the low probability of transition permits extremely lengthy runs, which we can compress quite easily into a few numbers. For example, a run of 100 zeros may occur, leading to a 100-fold reduction in the number of bits. However, the gain from the massive summarization of data quickly yields to the large number of possible run-lengths, spanning a number of nearly equally likely possibilities; the compression ratio quickly decreases to unity as the amount of randomness in the sheer number of possible run lengths (1 to 100+) that we must encode overtakes the savings gained from shrinking long runs. On the other end of the scale, for high transition probabilities $\alpha > 0.9$, the runs are short, and the bit alternations frequent, but the number of run-lengths we must encode also shrinks to a small typical set {mainly 1 and 2}, allowing us to describe the potential outcomes in fewer bits.

The probability mass functions for three representative values of α confirm our hypothesis:



The low probability ($\alpha = 0.05$) yields a large alphabet of possible run-lengths that we must encode, but the longer runs facilitate data reduction from many bits to a single representative count. The intermediate probability ($\alpha = 0.5$) balances alphabet size (≈ 16) with common lengths, but the mere possibility of longer runs forces us to accommodate them, and, alas, the savings gained from compression fails to exceed the amount of uncertainty in the large alphabet of run-lengths. The high probability ($\alpha = 0.95$) yields an easily encodable alphabet, thereby achieving a maximal compression ratio despite the rarity of actual "runs."

When we apply run-length encoding to the images *Leisler* and *Snoopy*, we observe that the resultant compression ratio depends on how we vectorize our two-dimensional image into a single

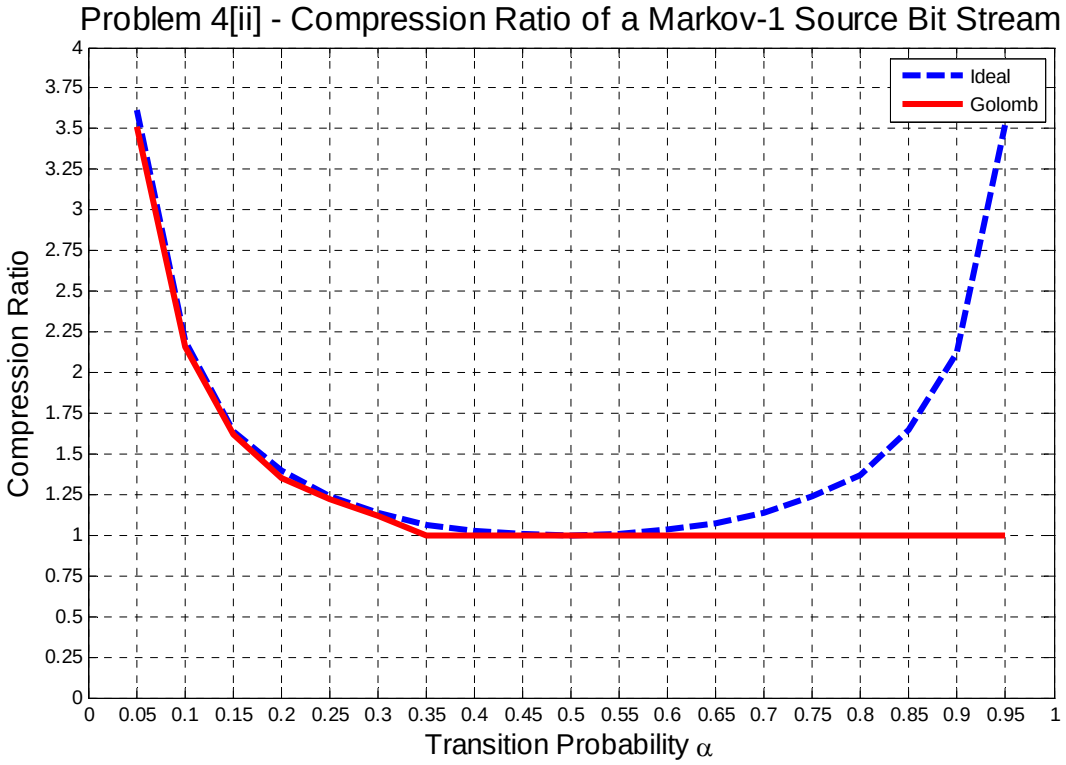
one-dimensional bit stream; unsurprisingly, the way we scan our image changes the distribution of runs, as the following table reveals:

IMAGE	**COLUMNWISE (Up → Down)**	**ROWWISE (Left → Right)**
Leisler	1.764204	1.780973
Snoopy	1.716713	1.844704

Thus, we can maximize our compression ratio by juxtaposing the image and its transposition to determine how one might exploit the distribution of runs for the best compression. For the *Snoopy* image, for instance, the inclusion of text in the transcript bubble creates several runs (lines) of white pixels, allowing us to exploit their redundancy in run-length coding; armed with this prescience, we scan left to right to encounter these white lines and maximize the length of our runs.

Problem #4 – Adaptive Golomb Coder

If we compress a Markov-1 source sequence with Golomb-coded run-length encoding, then we can achieve different compression ratios since the Golomb code should optimal for geometric sources. However, it remains to be seen whether or not our Golomb code can compete with the ideal entropy coding. We juxtapose Golomb compression ratios with ideal entropy coding:



The Golomb code compresses the run-length bit stream admirably well for Markov-1 source probabilities $\alpha < 1$, its compression ratio essentially coinciding with the ideal optimum computed as binary entropy. However, for higher transition probabilities – more state alternation, shorter and fewer runs – the Golomb code breaks down, failing to compress the data at all; most likely, this shortcoming unfolds because the data grow less and less geometric as the state transitions (bit flips) occur more frequently. With fewer runs to exploit in unary coding, the Golomb code mirrors the complexity and length of the run-length sequence itself.

Solid initial estimates for A and N_{max} materialize from repeated trial and error. In particular, A represents the average run-length (or most frequently Golomb-encoded value); hence, based on our observation from the histograms for α = 0.05, 0.5, and 0.95, we might select A of 4 or 5, since these run lengths occur quite frequently for intermediate alpha. Of course, Golomb code targeting lower values of α benefit from higher values of A since the possible run lengths stretch past 100; conversely, higher values of α demand A of unity, as runs hardly ever occur for frequently transitioning source codes. Meanwhile, because modern technology makes memory an afterthought – especially for the simple images we compress here – we need not worry about limiting the amount of hindsight our coder taps; instead, we choose N_{max} well above the number of elements in our system, ensuring that our Golomb coding algorithm considers the entire input stream.

Compressing the *Leisler* and *Snoopy* images, we obtain the following compression ratios:

IMAGE	**COLUMNWISE (GOLOMB)**	**COLUMNWISE (IDEAL)**	**ROWWISE (GOLOMB)**	**ROWWISE (IDEAL)**
Leisler	1.400748	**1.764204**	1.386630	**1.780973**
Snoopy	1.525938	**1.716713**	1.454416	**1.844704**

Juxtaposition with the ideal entropies reveals that the Golomb coder performs quite far from the optimum for *Leisler* and *Snoopy* images. Whereas the run-length codes could be compressed as much as 1.7 with ideal coding, the Golomb coding can achieve only 1.4 for the *Leisler* image and approximately 1.5 for the *Snoopy* image. The ideal entropy coder prevails over the Golomb coder.