

Data Compression with Huffman codes

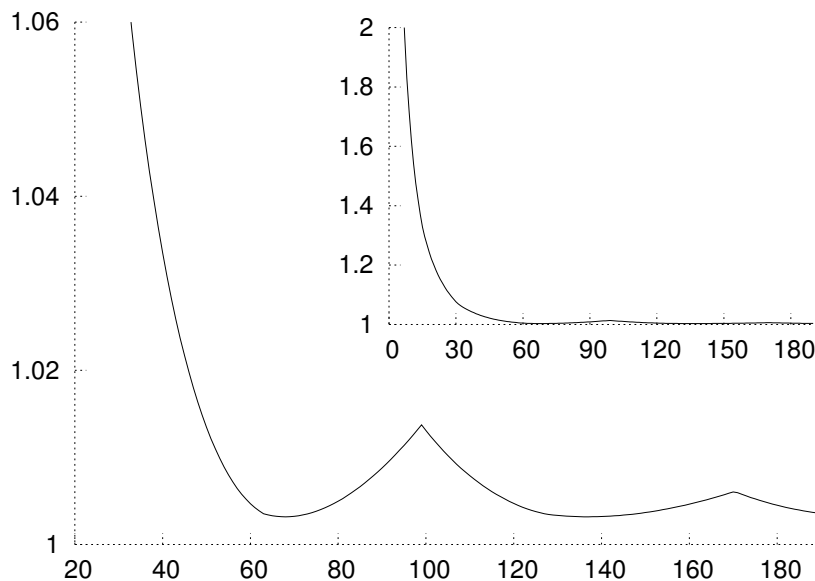
Consider a binary file that is very sparse – only $f = 1\%$ of the bits are 1s. How can we compress it into a smaller file? And how much compression should we expect is possible?

One compression method is to count the number of runs of zeroes between each successive pair of ones. Then we can encode these outcomes $\{r\}$, the run-lengths, using an optimal symbol code.

In principle, any one run of zeroes could have a very long length. Rather than assume really long runs will not happen, we would prefer to have a well-defined method for handling long runs. One idea is to pick a maximum run length of zeroes (without any one) and devote one of the leaves of our Huffman tree to encoding pure zeroes.

What should we choose as this maximum run length? Any value will give a valid code, but we are interested in minimizing the expected length L .

As we vary the maximum run length, r_{\max} , the ratio of the expected length to the entropy, L/H , varies as shown in the figure below.



If we use a maximum runlength greater than 40, then we get within 2% of perfect compression. But the efficiency doesn't decrease monotonically with r_{\max} . The efficiency is closest to 1 when $r_{\max} \simeq 69, 2 \times 69, \dots$. Can you see why? [Hints: what is the value of 0.99^{69} ? $\ln 2 \simeq 0.69$.]

For much larger values of r_{\max} , does Huffman coding do any better?

Source code implementing this compressor and other compression algorithms is available at:

<http://www.inference.phy.cam.ac.uk/mackay/itprnn/code/c/compress/>

This is an addition to *Information Theory, Inference, and Learning Algorithms* (Cambridge Univ. Press, 2003), which is available online from:

<http://www.inference.phy.cam.ac.uk/mackay/itila/>